



ザ8086ブック

[新装版]

16ビット・マイクロプロセッサ8086・8088の使い方

ラッセル・レクター／ジョージ・アレクシー共著 吉川敏則訳

秋葉出版



本書の特色

●ハードウェア

8086チップの構成，タイミング，さらに設計について詳細に解説。

●プログラミング

適切なプログラミング技法の解説とともに，完全な8086命令セットを示す。

●インターフェイス

すべての種類の素子のインターフェイスについての技法と仕様の解説。

●アプリケーション

8086の特殊な特徴を客観的に示すと共に，MultibusとマルチCPUの構成についても詳述。

1987年4月25日
有隣堂にて





ザ 8086ブック

16ビット・マイクロプロセッサ8086・8088の使い方

ラッセル・レクター／ジョージ・アレクシー 共著 吉川敏則 訳



秋葉出版

The 8086 Book

by Russell Rector / George Alexy

© 1980 by OSBORNE / McGraw-Hill, Inc.
Japanese Translation Rights Arranged with
McGraw-Hill International Book Company,
through Japan UNI Agency, Inc.

著者の序文

この本は、一般的なプログラミングの概念と実施、8086マイクロプロセッサとそのアセンブリ言語、8086マイクロプロセッサを用いたロジック・デザインの3つの話題に焦点を置いている。一般的なプログラミングの概念と実施の議論は、どのマイクロプロセッサにも関連している。しかし、この本の残りは8086に特有の部分である。それだけで、この本は8086を用いるためのテキストとなる。

8086マイクロプロセッサのプライム・ソースは、次のものである。

INTEL CORPORATION

3065 Bowers Avenue

Santa Clara, California 95051

一般的なプログラミングの概念と実施の解説は、プログラマとコンピュータとの間の関係を調べることから始まる。というのは、これが結局はデザイン・プロジェクトの本質を決定するものだからである。他のプログラマがアセンブリ言語あるいはより高いレベルの言語でプログラミングを行なっているときに、どうして機械語を用いて行なうプログラマがいるだろうか。異なるタイプの応用には異なるタイプのプログラミングが要求される。いずれの場合においても、良いプログラミングの実施は洗練される必要がある。この目的を成就するために、一連の規則について述べ、プログラミングのプロジェクトを説明するために2つの例を用いる。

8086マイクロプロセッサ自身の記述は、アセンブリ言語のプログラミングとハードウェアの設計を含んでいる。

アセンブリ言語のプログラマに対しては、8086のCPU構成とマイクロプロセッサのアセンブリ言語の命令セットについて、詳細に述べている。

ハードウェアの設計者に対しては、一般にマイクロプロセッサへ入力されるかあるいは、マイクロプロセッサから出力されるすべての信号に関するタイミングとバスの要点について述べている。シングルバスとマルチバスの構成も含まれている。標準のIntel Multibusについても詳細に記述してある。

この本が読者に仮定している知識

An Introduction to Microcomputers: Volume 1 - Basic Concepts, 2nd Revision, by A. Osborne, Osborne/McGraw-Hill, 1980 に述べられている一般的なマイクロプロセ

ッサの概念の実用的な知識や考え方を読者がもっていることを、この本では仮定している。したがって、2進数の算術演算、バッファ、あるいはCPU構成の原理などの基本的要素は、この本に含まれていない。

8086マイクロプロセッサとその直接のサポートの部分については、この本で詳細に取り上げてある。8089 I/O プロセッサにも触れているが、詳しくは述べていない。この詳細な記述については、The 8089 I/O Processor Handbook, by A. Osborne, Osborne/McGraw-Hill, 1980 を参照されたい。

日本語版刊行に際して

1970年代初期に紹介されて以来、マイクロプロセッサが日常生活の中で占める割合は、ますます大きくなってきている。自動車を運転するときも、ゲームマシンで遊ぶときも、われわれは、この強力なプロセッサとインターフェイスし、利用しているわけである。コンピュータをベースにした製品なくしては、各人が日常生活をおくれなくなる日も近いであろう。このような製品を使うことによって、人類は単純労働から解放され、社会を改善するための創造的な活動に従事することができるようになる。

新世代のマイクロプロセッサ技術が出現するたびに、このゴールが近くなってくる。最新の進歩といえば、16ビット・マイクロプロセッサの出現であろう。16ビット・マイクロプロセッサは、従来の8ビット・マイクロプロセッサに比べて、処理速度とデータ量が数倍になっている。この世代の中で最もポピュラーなCPUは8086で、前世代の8ビット・マイクロプロセッサ8080に比べて、処理能力が10倍、メモリ・アドレス領域が16倍になっている。その高度なコンピュータ・アーキテクチャのために、8086ファミリーは、色々なアプリケーション分野で非常にポピュラーになっている。高速、高精度の浮動小数点演算という機能を8087演算プロセッサで追加したり、高度なI/Oコントロールを8089 I/Oプロセッサで追加したりできるという、そのユニークなアーキテクチャは、8086ファミリーを、さらに広範で、高度なアプリケーションに適合させた。

また、そのフレキシブルなアーキテクチャのゆえに、他のどの8ビット・プロセッサに比べても、2倍以上のパフォーマンスを有する8ビットの8088から、8086とソフトウェア互換性を保ちながら、6倍のパフォーマンスと、オンチップのメモリ・マネージメントとメモリ・プロテクション機能を有する286まで、そのファミリーは充実している。

特に重要なのは、8086がパーソナル・コンピュータと科学技術計算の分野で、非常に広範に受け入れられていることである。簡潔なデザインに起因する高いコスト・パフォーマンスと、10社近くあるセカンドソース、そして広範でしかも、さらに増加しつつあるソフトウェアの基盤により、8088と8086は新製品にどんどん採用されている。8088と8086が広く受け入れられているということは、次世代のコンピュータ製品を使う人々のほとんどだれもが、8088や8086ベースの製品を使うことになるであろうということである。したがって、8088と8086のハードウェアとソフトウェアの両面を完全に理解したい、という要求が生まれてくるわけである。

この8086ブックのゴールと目的は、8086のアーキテクチャとアセンブリ言語およびハー

ドウェア設計のテクニックを詳細に説明することによって、この要求を満たすことである。本書を使用することによって、学生あるいは8088、8086のユーザは、この2つのプロセッサに関する知識を、迅速かつ効率的に得ることができ、新たに得た知識をハードウェアとソフトウェアの開発に適用することができる。なお本書は、読者が一般的なマイクロプロセッサの概念を持っていることを前提としている。

本書を精読したなら、読者は世界標準である8088と8086をベースにしたシステムの設計と、プログラミングに参加する資格と自信ができたものと考えてよいだろう。

1982年7月

G. Alexy

訳者のまえがき

私が初めてコンピュータに直接触れることができたのは、今から10年以上前のことになる。当時としてはかなりの規模のコンピュータであったが、今としては時代遅れの機種となっている。それ以来、大型から小型まで、何種類かのコンピュータを使う機会があった。

FORTRANやPL/Iなどの高級言語を使用しなかった訳ではないが、私が初めから興味を持っていたのは、主としてアセンブリ言語であったと言える。その理由の1つは、自分自身でコンピュータを動かしてみたい、あるいは操作してみたいといった欲求に駆られたからかもしれない。そして、この欲求を満たしてくれたのは、大型あるいは中型のコンピュータではなく、小型のそれもミニコンと呼ばれる機種であった。中型以上の機種ではそのシステムが複雑で、ユーザの立ち入ることの可能な部分が限られ、あまり興味がわかなかった。あるいは自分の能力の限界を超えていたのかもしれない。

コンピュータは、たとえミニコンと呼ばれるものであっても、決して安価な装置とは言えない。ところが、私が初めてコンピュータに触れることができた頃、ミニコンよりもさらに小型の、マイクロコンピュータが作られていた。正確には、マイクロプロセッサと言うべきであろうが、これがエレクトロニクスの発達の波に乗り、非常な急成長を遂げて現在に至っている。最初は4ビットであったものが、8ビット、さらに16ビットへと発展し、ミニコンとの区別がつかないようにまでなりつつある。また、マイクロプロセッサ自体の価格は非常に安くなり、機器に組み込まれたり、あるいはマイコン（マイクロ・コンピュータ）として、広範囲にわたってマイクロプロセッサが用いられている。

16ビットのマイクロプロセッサは、まだ開発されてから日が浅いが、今後多くの分野で積極的に取り入れられていくことは容易に推測できる。勿論その価格が下がることも要因の1つであろうが、その機能や能力の大きさは非常に魅力がある。

この本には、主として8086について、そのハードウェアとアセンブリ言語による命令の解説が示されている。ただし、著者の序文にも書かれているように、基礎的知識を既に読者がもっていることを仮定しており、また本文中では8086のアセンブリ言語の概要については述べられないままに、プログラムの例が示されている。したがって、この本で初めてアセンブリ言語に接する場合には、多少戸惑いをいだくかもしれない。

また、コンピュータ関係の本に限ったことではないが、この種の本では文章中に専門用語が多く、初心者にはなじみにくいものとなりやすい。本書を訳す際には、専門分野で英語で用いられている用語はできるだけ英語のままで残し、多少でも用語の理解に役立つこ

とを考慮して、注釈としてその訳を付けるようにした。

コンピュータ関係の専門書に少しは慣れているものと思い、この本の翻訳を引き受けてしまったが、いざ始めると自分の英語の貧弱さをまざまざと見せつけられてしまう破目になり、結果的には読者の判断を仰ぐことになってしまった。この本に関して読者の率直な御指摘・御批判をいただければ幸いである。

最後に、翻訳の機会を与えていただいた東京工業大学の当麻喜弘教授ならびに産報出版株式会社の木内雄一氏に心から御礼を申し上げる。

昭和57年 7 月

吉川 敏則

訳 者 補 足

1980年 9 月以降、以下のように名称が変更されている。括弧の中は旧名称を示す。

iAPX86/10 (8086)

iAPX88/10 (8088)

iAPX86/20 (8086と8087のシステム)

iAPX88/20 (8088と8087のシステム)

また、iAPX86の拡張として、

iAPX186 [iAPX86にクロック・ジュネレータや DMA コントローラなどを組み合わせて 1 チップとしたもの]

iAPX286 [仮想メモリを採用し、論理アドレス空間 $2^{30} \div 1 \text{ G byte}$ 、物理アドレス空間 $2^{24} \div 16 \text{ M byte}$ を持つ、メモリ保護や特権レベルを有する]

などがある。

目 次

第1章 プログラミング	(1)	3.1.3 出力	34
1.1 アセンブリ言語	1	3.1.4 プログラム設計	35
1.2 プログラミングの作業	4	3.2 8086の命令セット	39
1.2.1 システムの仕様	5	3.3 8086のレジスタとフラグ	41
1.2.2 プログラム設計	8	3.3.1 汎用レジスタ	42
1.2.3 プログラムの実現	9	3.3.2 ポインタ・レジスタ	43
1.2.4 試 験	12	3.3.3 インデックス・レジスタ	43
1.2.5 文書作成	13	3.3.4 セグメント・レジスタ	43
1.2.6 メインテナンス	13	3.3.5 フラグ・レジスタ	44
		3.3.6 命令がどのようにフラグ・ レジスタに作用するか	46
第2章 プログラム例	(15)	3.4 8086のアドレッシング・モー ド	50
2.1 ソート・プログラム	15	3.4.1 プログラム・メモリ・ア ドレッシング・モード	51
2.1.1 入 力	17	3.4.2 データ・メモリ・アドレ ッシング・モード	52
2.1.2 計算処理	17	3.4.3 アドレッシング・モード ・バイト	58
2.1.3 入力レコードのフォーマ ット	17	3.4.4 セグメント変更	60
2.1.4 ソートの方法	18	3.4.5 メモリ・アドレッシング ・テーブル	61
2.1.5 出力レコードのフォーマ ット	19	3.5 命令セットのニーモニック	62
2.1.6 出 力	20	3.5.1 略 語	62
2.1.7 エラー処理	20	3.6 8086アセンブリ言語の命令 (アルファベット順)	66
2.1.8 プログラム設計	20	3.7 アセンブラ依存のニーモ ニック	262
第3章 8086アセンブリ言語の 命令セット	(25)		
3.1 I/O ドライバ	28		
3.1.1 入 力	30		
3.1.2 計算処理	34		

第4章 8086の命令グループ (265)

- 4.1 データ移動命令265
 - 4.1.1 バッファからバッファ
への移動ルーチン269
 - 4.1.2 CPUの状態の退避.....277
 - 4.1.3 セグメント・レジスタの
初期設定278
- 4.2 算術演算命令279
 - 4.2.1 加算命令279
 - 4.2.2 減算命令282
 - 4.2.3 乗算命令285
 - 4.2.4 除算命令288
 - 4.2.5 比較命令291
- 4.3 論理演算命令294
- 4.4 ストリング・プリミティブ
命令302
 - 4.4.1 REP プレフィックス.....304
- 4.5 プログラム・カウンタ制御
命令306
 - 4.5.1 条件付きジャンプ命令 ...309
- 4.6 プロセッサ制御命令310
- 4.7 入出力命令316
- 4.8 インタラプト命令318
- 4.9 ローテートとシフトの命令 ...320

第5章 ソフトウェア開発 (329)

- 5.1 エディタ331
 - 5.1.1 エディタの機能331
 - 5.1.2 システム・コマンド337
- 5.2 アセンブラ338
- 5.3 デバッガ340

第6章 8086アセンブリ言語の プログラミング例 (343)

- 6.1 ソート・プログラム343

6.2 I/O ドライバ351

第7章 8086マイクロプロセッサ (357)

- 7.1 8086 CPUのピンと信号357
 - 7.1.1 アドレスとデータの
ライン359
 - 7.1.2 コントロールとステータ
スのライン360
 - 7.1.3 パワーとタイミングの
ライン363
 - 7.1.4 スリーステートのライン
と信号364
- 7.2 8086の概要と基本的システム
の概念364
 - 7.2.1 8086バス・サイクルの
定義364
 - 7.2.2 8086のアドレスとデータ
・バスの概念367
 - 7.2.3 システム・データ・バス
の概念372
 - 7.2.4 8086のエグゼキューショ
ン・ユニットとバス・イン
ターフェイス・ユニット ...383
 - 7.2.5 8086命令キュー384

第8章 8086の基本デザイン (389)

- 8.1 動作モード389
 - 8.1.1 ミニマム・モード389
 - 8.1.2 マキシマム・モード389
- 8.2 クロックの発生397
- 8.3 リセット404
- 8.4 レディの実現とタイミング ...408
- 8.5 インタラプト構造413
 - 8.5.1 定義済インタラプト414
 - 8.5.2 ユーザ定義ソフトウェア
・インタラプト416

8.5.3 ユーザ定義ハードウェア ・インタラプト	416
8.5.4 インタラプト・アクノリ ッジ・シーケンス	416
8.5.5 システムのインタラプト 構成	422
8.6 8086バス・タイミング図の 解釈	426
8.7 ミニマム・モード・バスの タイミング	427
8.7.1 アドレスとALE	427
8.7.2 リード・サイクルのタイ ミング	427
8.7.3 ライト・サイクルのタイ ミング	428
8.7.4 インタラプト・アクノリ ッジのタイミング	429
8.7.5 レディのタイミング	430
8.7.6 バス・コントロール移動 のタイミング	431
8.8 マキシマム・モード・バスの タイミング	431
8.8.1 アドレスとALE	431
8.8.2 リード・サイクルのタイ ミング	432
8.8.3 ライト・サイクルのタイ ミング	433
8.8.4 インタラプト・アクノリ ッジのタイミング	433
8.8.5 レディのタイミング	434
8.8.6 その他の考察	435
8.9 バス・コントロールの移動 (HOLD/HLDAと $\overline{RQ/GT}$)	435
8.9.1 ミニマム・モード	435
8.9.2 マキシマム・モード ($\overline{RQ/GT}$)	440

第9章 Multibus (449)

9.1 初期化信号ライン	451
9.2 アドレスとインヒビットの ライン	451
9.3 データ・ライン	452
9.4 バス競合解決ライン	452
9.5 情報伝送プロトコルのライン	453
9.6 非同期インタラプト・ライン	454
9.7 パワー供給ライン	454
9.8 予備ライン	454
9.9 Multibus 構成の概念	459

第10章 8086のマルチプロセッサ構成 (463)

10.1 コープロセッサ	463
10.2 共有システム・バスにおける 多重処理	466
10.3 8289のバス・アクセスとレリ ーズ・オプション	475

付 録 (477)

A 8086命令セット一覧 —アルファベット順—	477
B 8086命令セット一覧—オブジェクト・ コード数値上昇順—	485
C 8086と8088ファミリーのAC, DC特性と信号波形	493
D 8088 CPUについて	543

索 引 (547)

著者の序文	Ⅲ
日本語版刊行に際して	V
訳者のまえがき	Ⅶ
目 次	Ⅸ

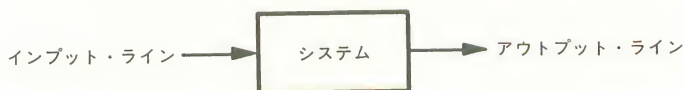
第1章 プログラミング



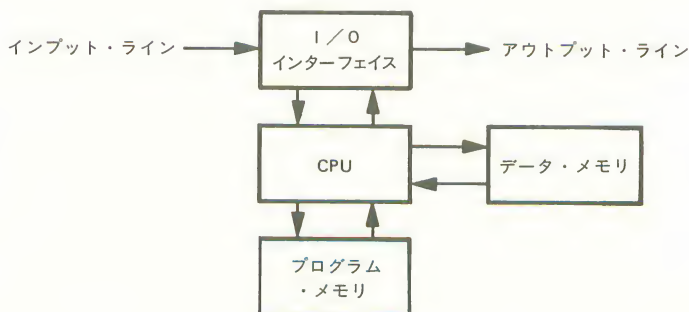
1.1 アセンブリ言語

マイクロコンピュータのシステムにおけるアセンブリ言語の機能は何か。機械語あるいはより高いレベルのプログラミング言語と、アセンブリ言語がどのように異なっているのか。この章では、アセンブリ言語が果たすいくつかの役割を評価することによって、これらの疑問に答える。

一般的な意味において、すべてのマイクロコンピュータのシステムは次の形式をとる。



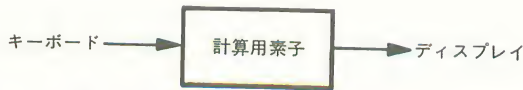
ここでインプット・ラインはシステムに情報を供給するために用いられ、アウトプット・ラインはシステムから情報を送るために用いられる。普通、システムは次のものから構成される。



CPU (Central Processing Unit) は、インプット・ラインから I/O インターフェイスを通してデータを受け取り、CPU はプログラム・メモリからの命令を実行することによって、このデータを処理する。結果は、アウトプット・ラインを通して出力される。CPU は、データ・メモリに一時的なデータをたくわえる。

CPU、I/O インターフェイス、そしてメモリは、システムのハードウェア部分であり、プログラム・メモリ中のデータは、システムあるいはプログラムのソフトウェア部分である。8086 アセンブリ言語の要素は、処理されてプログラム・メモリに格納されるプログラムを形成するために組み合わされる。したがって、アセンブリ言語は、メモリ中に存在するプログラムを記述するために用いられる。

プログラムの概念を理解するために、次の要素を持つ基本的 P O S (Point-Of-Sale) 端末を考えてみる。



キーが押されると、計算用素子は、押されたキーを機械が受け入れられるコードへ変換する操作を行なう。コードは処理されるべき数値あるいは実行されるべき計算を表わす。このコードを解釈することによって、計算用素子は必要な処理を実行し、ディスプレイ上に結果を表示する。

計算用素子は、一連のタスクを実行することによって、これらの処理を行なう。たとえば、計算用素子は、キーが押されたかどうかを判断するために、次の一連のタスクを実行する。

1. キーボードのステータス・バイトを読み取る。
2. ステータス・バイトからビット 3 を取り出す。

(ビット 3 が 0 ならば、キーは押されていない。ビット 3 が 1 ならば、キーは押されている。)

3. ビット 3 をテストする。

(ビット 3 が 0 ならばステップ 1 へ戻る。ビット 3 が 1 ならばステップ 4 へ進む。)

4. 次のタスクを実行する。これは、キーが押されたことを表わすビットをクリアするコマンド、あるいはキーボードの動作を無効にするコマンドである。

すべての変換と計算の処理を含む、計算用素子によって実行されるタスクの完全な集合は、アルゴリズムとして知られている。アルゴリズムは、開始点と終了の基準をもつ明確に順序づけられたタスクの系列より成る。アルゴリズムは普通、上述の例、すなわち実行すべきタスクを記述する文章の形式で表わされる。残念ながら、CPU は“キーボードのステータス・バイトを読み取る”のような文章に応答することはできない。文章で構成されているアルゴリズムから、CPU によって解釈可能な 2 進数の命令系列で構成されている形式への翻訳が存在しなければならない。アルゴリズムを実現するために用いられる命令の集合は、オブジェクト・プログラムとして知られている。

CPUは、2進数すなわち1あるいは0の数字より成る情報の単位を解析することによって、命令を実行する。簡単なCPUは、命令のフェッチと命令実行の2つのサイクルを持っている。命令のフェッチ・サイクルにおいてCPUは、実行されるべき次の命令（情報の単位）を含む位置のアドレスを生成し、メモリがその位置における情報の単位を供給することを要求する。メモリは適切な情報の供給を行なう。次の命令実行サイクルの間に、CPUはその情報を解析し、適当な動作を実行する。

たとえば、前の例で示したタスクを実行するために、インテルの8086システムにおいて、次のデータが与えられたと仮定する（アドレスと命令は2進数）。

アドレス	命令
0000	11100100
0001	00001010
0010	00100100
0011	00001000
0100	01110101
0101	11111010

8086が0000から実行を開始したとすると、0000における最初の命令を読み込んで、それを解析する。CPUは、その命令が入力命令であり、次の0001がデータの読み込まれるべきデバイスのアドレスを含んでいると判断する。したがって、デバイス・コードは00001010となる。もしデバイス・コード00001010のデバイスがキーボードのステータス・バイトを生成するならば、この命令の実行によってキーボードのステータス・バイトが8086のALレジスタに読み込まれる。0000の命令実行後に、次に実行される命令は0010の命令である。0010の命令は、ALレジスタとのANDを実行するために0011の情報をを用いる。これは、前述の例の2番目のタスクにおけるビット3の取り出しである。0100と0101の命令は、ビット3が1か0かを判断し、次いで適当な動作を行なう。

CPUは1と0を用いて動作する。しかし人は、1と0を用いることにそれほど慣れてはいない。したがって、人とCPUが用いている1と0の間に、中間のステップが設けられている。このステップが、アセンブリ言語である。コンピュータに1と0を直接入力する代わりに、人はアセンブリ言語でプログラムを書く。アセンブリ言語のプログラムは、アセンブラとして知られているプログラムによって適当な1と0に変換される。アセンブリ言語で書かれたユーザのプログラムは、ソース・プログラムと呼ばれている。

たとえば、前に示したように、1と0より成るプログラムを作成する代わりに、8086アセンブリ・コード（ソース・コード）の以下に示すラインがアセンブラに入力できる。

```
TOP:    IN      AL, 0AH
        AND     AL, 08H
        JNZ     TOP
```

アセンブラはこのコードを、前述の例の1と0（オブジェクト・コード）に変換する。アセンブリ言語は、アセンブラによってシステムが実行可能な1と0のすべての組合せに変換可能な命令の集合より成る。

たとえば、8086アセンブリ言語の命令

```
AND     AL, 08H
```


は、アセンブラによって次の2バイトのオブジェクト・コードに変換される。

```
00100100
00001000
```

このオブジェクト・コードは8086によって、ALレジスタの内容と00001000との間でANDをとる命令として解釈される。

いくつかの理由から、アセンブリ言語のプログラミングは、2進数コードによるプログラミングよりも効率が良い。第1に、01001000、10100010、あるいは01110000などの命令を書くよりも、AND、ADDまたはXORなどのアセンブリ言語の命令を用いてアセンブリ・コードを書くことの方が明らかに容易である。第2に、機械語による命令入力の際に、エラーの可能性は非常に高い。アセンブリ言語の作成の段階では、もしエラーが存在しても、それらは普通、アセンブラによって見つけられる。

1.2 プログラミングの作業

次に、プログラマとマイクロコンピュータ・システムの間関係について考える。マイクロコンピュータ・システムを働かせるために、プログラマが一般に行なう作業には以下のものがある。

- ① システムの仕様：システムが取り扱う入力と出力の特性の記述に加えて、システムが提供するすべての機能の一般的な解説が仕様に含まれる。
- ② 与えられたシステムについての仕様を実現するコンピュータ・プログラムの設計：これには、提案されたシステムが特定の応用に対処できる一連のステップとして、仕様が実現されることが必要とされる。
- ③ 特定のコンピュータ言語を用いたプログラム設計の実現：この段階には、コーディング、デバッグ、完成の3つの別々の作業が含まれている。
- ④ 完成したシステムのテスト：多くのテスト・データをシステムに入力する。プログラムのロジックとハードウェアの構成要素を働かせるように、テスト・データは設計される。
- ⑤ システムの文書作成：適切な文書作成には、全体のシステムがどのように動作するか記述、システムに対するオペレータの手引き、そしてプログラムの完全な文書化が要求される。
- ⑥ システムのメンテナンス：新しい要求あるいは新しい装置が必要とされるならば、システムを更新するためのプランが存在しなければならない。

複雑なシステムをプログラミングする場合、上記のリストは常に用いられる。しかし、システムの拡張に関する3つの小節を含む250ページの仕様と、50ページのオペレータの手引き、さらに厳密なテストの方法が必要とされない場合は限られている。これらの計画は、次のような種類の場合に現われる。

1. シリアル I/O チャンネルの動作が正常でない。ハードウェア関係者はソフトウェアを指摘し、ソフトウェア関係者はそのようなむさくしいものが常に正常であること

を信じるのは不可能だと考えている。有望なことにこの解決法は、チャンネルを初期化してシリアル I/O チャンネルがデータの有効性を示すごとにデータを読んで表示する短いプログラムにある。ハードウェア動作の確証を得ることは比較的容易であり、もしハードウェアが正常ならば、何が働いていないかについての疑問はほとんどなくなる。

2. 小数の重要な計算が行なわれる必要がある。幸運にも、FORTARN システムが利用可能である。有望なことに、希望する結果を生じるであろう20ステートメントの FORTRAN プログラムがその解となる。

前述のどちらの場合においても、仕様あるいはプログラムの計画が紙に書かれることは非常に少なく、これらのステップはプログラマの頭の中で行なわれる。これらの場合においては、おそらく文書が作られることはなく、メンテナンスの必要性は疑問となる。しかしこれらの例は、慣例に対して例外であることを覚えていることは非常に賢明である。

作業に関する①から⑥の項目は、多数のプログラマがいるような状況では、非常に有効に用いられる。何人かのプログラマはステップ①と②を専門に行ない、何人かのプログラマはプログラム設計の実現のみを行ない、何人かはプログラムのシステムのテストに時間のほとんどを費やし、他の人は文章やメンテナンスに活動を限定し、さらに他の人はプログラム作業の統括を行なう。このようにしてプログラマは、より高い生産性を可能とする特殊技能を磨いている。しかしながら、普通、アセンブリ言語のプログラマは、前述の作業のすべてを実行することを求められている。

この本では、アセンブリ言語による8086へのアプローチを重視している。そこで、これらの作業すべての一般的な解説を以下に示す。

1.2.1 システムの仕様

マイクロコンピュータの習得を最初に考慮すると、その必要性は次の2種類の解析の中から結果として生じる。

1. 直面する特殊な問題が存在している。たとえば、航空宇宙製造業者は、ある大きさと速度の必要条件を満たす搭載用計算システムを必要とするミサイル誘導システムを生産している。
2. 新しいマイクロコンピュータ・システムに対する特定の市場が存在する。たとえば、以前は会計のコンピュータ化の余裕のなかった小さな事業所は、マイクロコンピュータをベースとする業務システムの価格が十分に低くなれば購入する。

いずれの場合においても、計画されたシステムが果たすであろう正確な機能を明確にすることは重要である。

第1の場合において、問題の性質は、特定の機能を果たすためにおそらくシステムを制限する。第2の場合においては、仕様のみ考慮すればよい。小規模の業務システムについては、システムにおいてどのような会計の機能が実行されるかを明確に、そして各種のタイプのどれだけ多くの記録が許されるかを、正確に定義する必要がある。そうでなければ、

マイクロコンピュータのシステムは、そのハードウェアが取り扱うことのできる能力以上のタスクを割り当てられることになる。本章の初めのマイクロコンピュータ・システムの簡単なモデルを振り返ると、仕様は次のように定義される。

- ・システムによって受け取られる入力。
- ・システムによって実行される計算処理。
- ・システムによって作り出される出力。

(1) 入 力

マイクロコンピュータ・システムの入力の仕様は、実行されるプログラミングのレベルに大きく依存している。BASICを用いているアプリケーションのプログラマは、ディスク・コントローラに与えられるコマンドのタイプへの関心はあまりありそうにない。むしろ、ディスク上のデータのタイプ、ディスク・ファイルに記録がどのように割り付けられるか、ディスク・ファイルの操作をオペレーティング・システムがどのように圧縮するかなどに関心を持っている。この本自体はアセンブリ言語のプログラミングについて記述しているが、データ・ベースの取り扱いの技法についての適当な解説はこの本の範囲を越えているので、ハードウェア・レベルでの入力と出力について重点的に述べる。

ハードウェア・レベルで、3つのパラメータが入力チャンネルの特性を定めている。それは次のものである。

1. データ・バスの幅：プロセッサ・コントローラ・エラー・システムからは一度に1ビットの入力が到着する。パラレルまたはシリアルI/Oチャンネルは、一度に8ビットを入力する。フロッピー・ディスク・コントローラは、リクエストによって1024 (128 バイト) の情報を伝送する。
2. データ伝送の速度とタイプ (同期あるいは非同期)：リアルタイム・クロックの 200 マイクロ秒ごとに、データが到着する。10ミリ秒ごとに、シリアルI/Oチャンネルは非同期にデータを入力する。コントロール・システムのA/Dコンバータは未定の速度でデータを伝送するが、速くても500ミリ秒に1回の速度である。
3. 付随のコントロール情報：データが有効になると、フロッピー・ディスクは割り込みを発生する。キーボード・サブシステムは、データが有効ならばステータス・ビットをセットする。新しいデータが有効かどうかを決定するために、システムが入力を読み取り、前のデータと比較することをA/Dコンバータは要求する。

これらのパラメータが説明された後で、入力チャンネルがどのように動作するかを述べることは重要である。

入力チャンネルには普通、以下に記す3つのタイプのポートがある。

1. データ・ポート：このポートは、システムの処理部分へ送られるデータを含んでいる。
2. ステータス・ポート：このポートは、いつデータが有効か、このチャンネルにエラーが発生していないかを示す情報と外部に関する情報を含んでいる。
3. コントロール・ポート：このポートは一般に、チャンネルの動作モードを初期化するためと、チャンネルの外部に対する表示方法をコントロールするために用いられる。

この3つのポートは常に存在するとは限らない。ある場合には、データ・ポートだけが存在し、ある場合には、パワー供給時にチャンネルが自動的に初期化され、したがってコントロール・ポートを必要とはしない。

(2) 計算処理

マイクロコンピュータ・システムの計算部分を述べるとき、以下の関連ある3つの主要な領域が存在する。

1. 入力部分からの生のデータの処理：これは、データのブロックをその構成部分（たとえば、ディスクからの1セクタのデータを、ファイル・ヘッダ、ヘッダ・チェックサム、データ、そしてデータ・チェックサム）に分離して、システムによってより容易に利用できるコードへの変換（たとえば、ASCII から2進数へ）の形を取ることができる。
2. システムによって実行される実際のアルゴリズム：実際のアルゴリズムの完全な記述は普通、プログラム設計でなされ、仕様のこの部分ではシステムが実行する主要な機能のすべてを記載しなければならない。
3. 出力部分のデータ処理：この処理には、出力装置によって利用可能な形式へのデータの変換（たとえば、2進数データの EBCDIC への変換）が含まれる。

(3) 出力

マイクロコンピュータ・システムの出力の記述は、入力部分で行なわれたものと非常に類似した解析を必要とする。各出力チャンネルについては、以下の3つのパラメータが存在する。

1. チャンネルによって伝送されるビット数。
2. 出力チャンネルにおけるデータ伝送速度。
3. いつトランスミッタがデータを要求するか、あるいはさらにデータを取り扱うことができるかを、システムに知らせる付随のコントロール情報。

これらのパラメータの説明に続いて、チャンネルがどのようにコントロールされるかを述べる必要がある。入力チャンネルと同じく、出力チャンネルは普通、以下に示す重要な3つのポートを持っている。

1. データ・ポート：このポートは外部へ伝送すべきデータを受け取る。
2. ステータス・ポート：このポートは、いつデータがデータ・ポートに伝送されるか、チャンネルでエラーが発生していないかを表わす情報と、外部に関する他の情報を含んでいる。
3. コントロール・ポート：このポートは一般に、チャンネルの動作モードを初期化するためと、チャンネルの外部に対する表現方法をコントロールするために用いられる。入力チャンネルと同様に、これらのポートのすべてが出力チャンネルをコントロールするために必要ではない。

3つの主要な各部分についての記述の過程で、以下に示す覚えておくべきいくつかの有用な技法が存在する。

1. 各部分において、発生する可能性のあるエラーの状態やエラーに対するシステムの

応答の一覧表の作成。

2. 各部分において、その部分が取り扱うすべての機能の一覧表の作成。たとえば、すべての入力チャンネル、すべての計算処理機能、さらにすべての出力チャンネルの一覧表の作成。特定の部分が終わったならば、システムに対するすべての可能性が認識されたことを十分確実とするように、その部分を一覧表で相互に照合する。

書かれている最初の仕様が必ずしも最終のものではない。もし手元にある問題があまり簡単なものでないならば、それはほとんど確実に最終のものとはならない。プログラム設計の作業と実施の仕事は、選択されたハードウェア構成が与えられたのでは、確実な機能が実行されるのは不可能なことを示す。この場合には、ハードウェア構成が変えられるように仕様を修正するか、あるいは与えられたハードウェア構成が実行できるように、障害となっている機能を変更する必要がある。

1.2.2 プログラム設計

プログラム設計には、仕様中のことばを解釈して、仕様を実現する方法を示す一連の言語でステップを書くことが含まれている。理想的には、この言語によるステップは、システムが成し遂げるべきことの明白で簡単な記述を与える。この時点で、すべてのシステムに対して簡単な記述が得られることは、直ちに明らかとはならない。たとえば、IBMのDOS/VSオペレーティング・システムの簡潔な記述を見出すことは期待されない。全体のシステムについてこのことは真実であろうが、理想的立場ではシステムの個々の部分（たとえば、プリンタ・ドライバや複数ワードの減算ルーチン）に対しては、簡単な記述が利用可能であるべきである。非常に大きいシステムに含まれる部分の数を考えると、規模の大きい仕様を多数のずっと小さいモジュールに分割するプログラム設計者の仕事がかすかに頭に浮かぶ。

プログラム設計の仕事に従事するときは、以下の提案を心に留めておかれたい。

1. 将来において、より大きい能力を備えるためにプログラムを拡張しなければならない。したがって、プログラムは組み込みによる拡張の機能を備えていなければならない。このような機能には、システム・サブルーチン、データの拡張可能なテーブルやリスト、システムにより多くの機能を付加するための便利でよく説明された方法、そして適度に融通性のあるデータ構造が含まれている。
2. 一般的な設計として、与えられた機能を果たすためには1つ以上の方法がある。ある場合には、機械の制限から1つの解法を用いることが強いられる。他の場合には、時間的な制約から別の解法が強いられる。これらの要因、すなわち機械と時間の制限は、実際のコーディングが行なわれる実施作業まで十分知られていないので、設計の段階ではしばしば特殊な問題を解く別の方法を追求することが賢明である。この段階で、他に代わるものを見出すことの利点は2つある。第1に、もし引用した制限が1つの解法を妨げるならば、もう一方は既に利用可能となっており、第2に、その過程でより有効な解法を発見するかもしれない。
3. 設計を行なっている間、特定のモジュールが他のモジュールにどのような影響を与

えるかを明白に記すことは非常に重要であり、またそのモジュールに他のモジュールがどのような影響を与えるかを明白に記すことも同じく重要である。モジュール間のこのインターフェイスは、プログラム・モジュールのデバッグと完成時に重要となる。プログラム設計が完成すれば、仕様を再検討するためにその設計を用いる。仕様と設計との相互の照合によって、設計や仕様における欠陥や手落ちが明らかにされる。正規の基準から設計が再検討されるべきであるという事実を自覚しなければならない。実現と試験の作業が行なわれている間に、プログラム設計の再評価を行なわなければならないような新しい情報が得られる。

1.2.3 プログラムの実現

プログラム実現の作業は、プログラム設計の作業で記述された言語のアルゴリズムを解釈して、特定のマイクロコンピュータ・システムで動作させることから成る。実現の作業に入るには、次の2つの異なる苦勞が存在する。

1. コーディング。これは、プログラム設計の作業で作成された言語によるステップを、特別なコンピュータ言語に変換する過程である。
2. デバッグと完成。これは、コーディングの段階でコンピュータ言語に変換されたプログラム設計モジュールからエラーを取り除き、このモジュールを動作するシステムに完成させる過程である。

(1) コーディング

プログラム設計を特別なコンピュータ言語に変換することは、プログラムの簡単な作業の一つとすることができる。もしプログラム設計の機能が正確になされたならば、個々のモジュールは1組の簡潔な文章によって記述される。コーディングでは、次の提案を心に留めておかれたい。

1. 可能ならば常に標準のサブルーチンあるいはプログラムを用いるように心がける。サブルーチンは通例、個々にデバッグすることができるという点で非常に有用である。サブルーチンからバグを取り除いた後は、コードのメインをデバッグすることはずっと容易となる。さらに、標準的なサブルーチンは、システムに新しい特徴を付加することを非常に容易なものとしている。
2. コードの文書をできるだけ明確にする。コードの個々のモジュールあるいはセクションについて述べる注釈文に加えて、覚えやすい意味を持つラベルはきわめて重要な価値がある。ラベルの文字数を6あるいはそれ以下に制限しているような、上記の機会を制限するアセンブラがある。しかしほとんどの場合、プログラムとデータ領域の両方のラベルに驚くべき記憶を助ける価値を与える能力が存在しており、十分に生かされるべきである。

各々のプログラム設計のモジュールが適当なコンピュータ言語に変換された後で、コーディングの作業が正確に行なわれていることを確実にし、デバッグの作業を行なう間、可能性のある障害を避けるために、一連の検査が行なわれる。この検査は、ときにはデスク・チェックングと呼ばれ、コーディング手順の一部分であるが、またデバッグの作業と多

くの要素を共有している。

行なわれる検査には以下のものが含まれる。

1. コードがプログラム設計のモジュールをすべて含んでいることの確認。
2. プログラム設計に含まれているすべての判断がコードに含まれていることの確認。
正しく分岐が行なわれることを確実にするための、すべての判断点における論理の検査。
3. 各々のプログラム設計のモジュールに、それを正しく動作させるために十分な情報が供給されていることの確認。この検査は、各モジュールについて、このモジュールが他のモジュールに供給することを要求するものが何かを、次の点から決定することによって行なわれる。
 - レジスタの内容
 - データ構造の内容
 - このモジュールによって用いられる I/O 装置の状態
 - ステータスの設定
4. 各モジュールが後続のモジュールに正確な情報を供給することの確認。この検査は、このモジュールが他のモジュールに供給すべきものが何かを、次の点から決定することによって行なわれる。
 - レジスタの内容
 - データ構造の内容
 - 後続のモジュールが用いる I/O 装置の状態
 - ステータスの設定
5. 次の状況を取り扱うためのコードがこのモジュールに記入されていることの確認。
 - エラー
 - 特殊な場合
 - 限界の場合
 - 通常の場合

これらの検査が終了した後に、デバッグの手順が始まる。

(2) デバッグと完成

デバッグと完成の作業は、コードからエラーを除去し、最終的に動作するシステムに、デバッグされたモジュールを完成させることからなる。デバッグの作業中に行なわれる機能は、机上の検査で行なわれる機能によく似ている。デバッグの作業は、システムのハードウェアあるいはシステムのハードウェアのシミュレータで、動作しているコードを調べる間にこの作業が行なわれる点で異なっている。机上の検査では利用できない、デバッグの過程で用いられる一連の道具がある。この道具は普通、常にではないが、デバッグと呼ばれるソフトウェア・モジュールによって提供される。デバッグによって与えられる代表的な特徴には、次のものが含まれる。

- シングル・ステップの機能：この機能は、プログラムのロジックに従う個々の命令の実行を、ユーザに可能とする。

- ・メモリあるいはレジスタの内容の検討・変更：この機能は、ユーザがメモリやレジスタの内容を調べ、任意に変更することを可能とする。
- ・ブレイクポイントの機能：この機能は、ある条件に従ってデバッグされているプログラムの実行を、ユーザが中断することを可能にする。代表的なブレイクポイントの条件には、オペランドの参照あるいは命令のフェッチに対する特定のアドレスの参照が含まれる。

プログラムをデバッグするときは、次の提案を心に留めておかれたい。

1. デバッグの過程は、共通に用いられているサブルーチンか、あるいはシステムのサブルーチンのデバッグから始める。もしソフトウェア・システムにおいて最下位レベルのルーチンが適切に機能していることが知られているならば、メインのコードが間違っているか、あるいは誤った方法でシステム・サブルーチンが用いられていると仮定することが可能なように、エラーの原因を発見することは簡単になる。
2. もし可能ならば、個々の仕様の各範囲のデバッグを試みる。個々に仕様の入力部分の各セクション、次いで仕様の計算処理部分の各セクション、次いで仕様の出力部分の各セクションをデバッグするのが適切である。仕様のセクションを別々にデバッグすれば、システムの他の部分からの干渉を受けずに各セクションを調べることができる。理論的には、個々のモジュールのすべてがデバッグされれば、完成の段階ではプログラムのモジュールが相互にインターフェイスするようにデバッグすることが必要となるだけである。

個々のモジュールのすべてがデバッグされれば、完成の段階に入る。この段階では、個々のモジュールがサブシステムに結合され、サブシステムとしてデバッグされる。たとえば、プログラムの入力部分に影響するプログラム設計のモジュールのすべてが結合されてデバッグされる。各サブシステムがデバッグされると、最終的なシステムがデバッグされるまで、他のサブシステムと結合が行なわれる。前に示したように、完成の段階で行なわれなければならない機能は、モジュール（結局はサブシステム）間のインターフェイスが正確に処理されていることの確認だけである。

完成のいずれの段階においても、プログラム設計あるいは仕様作成の作業にまで戻ることが必要となる場合がある。次の例を考える。

1. コーディングの段階で、指定された機能を設けるために必要なコードが、ハードウェアの設計で与えられた以上のメモリを要することが明らかになる。まず、他の方法でより少ないメモリ領域の利用が可能かを決めるために、プログラム設計の作業へ戻る。もしこれでも問題が解決しなければ、どうやら仕様作成の作業に戻り、システムを何かの方法で再構成すべきときである。
2. デバッグの段階で、システムがコントロールすることになっているすべての装置を動作させる試みが行なわれたときに、十分に速いシステムの応答が得られないことが認められる。仕様の入力と出力の部分は完成するまでは一般に別々にデバッグされるので、デバッグの初期の段階ではこの支障は明白にならないかもしれないことに注意。この場合、入力あるいは出力のコードの実行時間が減少しないかを調べるために、コ

ーディングの作業に戻る。もしこれがだめならば、もっと能率の良いアルゴリズムが利用できるかを判断するために、プログラム設計の作業へ戻る。まったくもってこれがだめならば、システムを改革するために仕様作成の段階に戻る。

1.2.4 試 験

試験の作業は、特殊なデータの集合を用いてシステムを十分に動作させて、正しい結果の得られることを確かめることである。この作業は、非常に多くのプログラマが存在する環境ではごく普通のことである。たとえば、自尊心のあるソフトウェア・ハウスがオペレーティング・システムの新しいバージョンを発売する前には、その新しいシステムは十分に吟味されている。自動車製造業者がコンピュータ・システム搭載のバージョンを発売する前には、厳格な試験が実施される。しかし、アセンブリ言語のプログラマが少ない状況では、試験はしばしば見落とされる。試験を無視する主な理由は、それが非常に時間を消費し、その結果非常に経費がかかることにある。その上、試験はよく理解された技術ではない。

有望なことに、試験の作業の重要な部分は実現の作業のデバッグ段階の間に行なうことができる。たとえばデバッグの部分で、境界条件となるデータを用いてモジュールを実行し、それによって判断すべきモジュールの機能を働かせて各モジュールが試験される。例として、データ・ブロックの最初のバイトが 30_{16} から 39_{16} までの範囲で1つの機能、最初のバイトが 41_{16} から 46_{16} までの範囲でもう1つの機能、そして最初のバイトがどちらの範囲でもない場合に第3の機能を実行するためのモジュールのコード化を考える。代表的な試験データは、次の値を最初のバイトに持つブロックを適当に含んでいる。

$2F_{16}$
 30_{16}
 39_{16}
 $3A_{16}$
 40_{16}
 41_{16}
 46_{16}
 47_{16}
 00_{16}
 FF_{16}

これらのブロックは、データの異なるタイプを区別するシステムの能力を試験する。システムに従った試験データを決定するときは、次の提案に留意されたい。

1. 入れるべきデータには次の3つの基本的なタイプがある。
 - ・システムが通常出会うデータの典型的な流れ。
 - ・判断を正確に実行するシステムの能力を働かせる一連の境界条件。
 - ・論理的データと非論理的データの両方を含むデータのランダムな選択。
2. データは次の速度でシステムに与えられなければならない。

- ・システムが通常出会う典型的なデータ速度。
- ・システムが機能するとされている最高のデータ速度。
- ・データ速度のランダムな選択。

1.2.5 文書作成

文書作成の作業は、システムに関するすべての情報を書き留めることである。システムの文書作成には、次の3つの基本的な要素がある。

1. プログラムに関する文書作成：実際の作業の解説のところで示したように、モジュールごとに、コードがどのように働き、ある場合にはなぜコードがそのように動作するかを説明することは非常に重要である。この種類の文書作成は、コードの新しい読者にもコードと容易になじむことを可能とする。さらに、コードを変更する必要があるが生じて、変更がどのように行なわれるかの合理的で精通した判断が、すぐれた文書によって可能となる。プログラムに関する文書は、ずっと以前に書かれたプログラムを調べたり、最初にコード化を行なった人の記憶を新たにする手助けとなる。
2. システムの手引き：システムの手引きには、プログラム設計の記述、プログラム変更方法の記述。システムが外部で何が起きることを予期しているか、すなわち、何が入力ラインを駆動し、何が出力ラインでデータを受け取るかの簡潔な要約が含まれるべきである。有望なことに、システムの手引きは、以前の作業の間に少なくとも文書の形で主要な要素が詳細に書かれているはずなので、かなり簡単にまとめられる。
3. ユーザの手引き：これは文書の中で最も重要な部分である。もしコードに関する文書がよく整っていて、さらに最も良いシステムの手引きが書かれていたならば、他のプログラマはプログラムを変更あるいは改良することが可能となる。しかし、もしユーザの手引きがなければ、だれもプログラムを使用することはできず、この場合、コードを変更あるいは改良しようとする人はいなくなり、すべての努力が無駄となる。システムとインターフェイスするプログラムを外部のユーザが書くようなシステムでは、ユーザの手引きは特に重要である。この場合、システムの改訂あるいは付加は、典型的にはユーザの手引きの更新を通して、ユーザに知らせることが最も重要である。

1.2.6 メインテナンス

メインテナンスの作業は、新しい設備あるいは新しい処理要求に適應するためのプログラム変更より成り、本質的には変化する環境でプログラムの機能を保つことにある。メインテナンスの作業は、環境の変化に応じて、簡単にも複雑にもなる。簡単な作業の例には次の事項が含まれる。

1. 装備の時代遅れの部品に代わって、システムにハードウェアの新しい部品が取り付けられる。I/Oインターフェイスは古いハードウェアと非常に似ていて、実際、変更はいくつかのステータス・ラインの入れ替えだけを含んでいる。この場合、メインテナンスの作業には、プログラムの数行のコードの変更、新しいハードウェアを用いたこのコードのデバッグ、さらに文書への適当な付加事項の記録が含まれる。

2. いつか後でより複雑なシステムによって処理される可能性のあるディスク・ファイルに、システムが出力を行なっている。より複雑なシステムの要求によって新しいオペレーティング・システムが作られ、重要なある目的のために、ディスク・ファイルで以前使われていなかった2バイトが用いられる。この可能性はプログラム設計の段階で考慮されるので、この場合のメンテナンスの作業は、仕様決定作業の計算処理の部分に関連したプログラムの設計、実現、さらに文書作成の作業に小さい変更を要するだけである。

より複雑な作業には次のものが含まれる。

1. ハードウェアの新しい部品がシステムに付加される。前の例と比較して、この装置は他のシステムの装置と少しも類似性がなく、実際、インタラプト構成、システムのタイミング、さらにシステムの処理能力について新しい要求が出される。この場合、メンテナンスの作業は確認されているプログラミングの作業の各々において広範囲にわたる労力を必要とする。
2. 販売部門で、マイクロプロセッサに基づくシステムに80メガバイト・ディスク・ドライブの実集合体を付加することを決定する。この場合のメンテナンスの作業には、仕様から文書作成までのプログラミングの作業のすべてが多分含まれるだろうし、あるいは想像できるように能力集団から適当な販売人員の順序づけられた配置換えを含むことになる。

比較的容易にメンテナンスの作業を行なう機能は、プログラム設計と実現の段階で払われた配慮に直接に比例する。プログラム設計の段階で特徴を付加する容易な方法が残されていないか、あるいは一般的システムのモジュールが与えられていなければ、どんなによくみてもシステムへの付加は困難であることが多分証明される。プログラム実現の過程で、プログラムの方法と理由についての合理的な文書が供給されていなければ、コードにプログラム設計の新しい要素を導入することはまったく遠回りなものとなる。

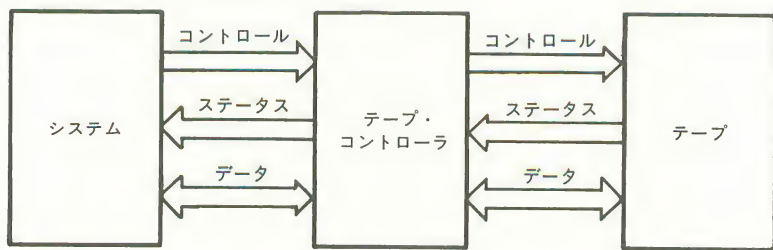
第2章

プログラム例

2.1 ソート・プログラム

ソート・プログラムのモジュールの仕様決定とプログラム設計の作業について考える。このプログラムは、テープ・ドライブのファイルからデータ・レコードを読み取り、レコードから分類キーを取り出し、テープにデータ・ファイルに続いてキー・ファイルを書き込む。このプログラムの実際のコードは、第6章に示す。

この例では、非常に簡単なI/Oインターフェイスを仮定する。I/Oインターフェイスに対する一般的なブロック図を次に示す。



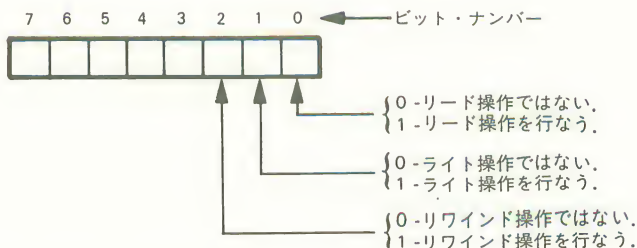
テープ・コントローラは、テープに対して128バイト・ブロックのデータを伝送する。コントローラは、テープに書き込まれるブロックに、パリティ・ビット（9トラックのテープに対して）とチェックサムを付加する。コントローラは、読み込み動作が行なわれるときにこのエラー検出の情報を処理し、それに応じてエラー・ビットを設定する。

テープとの間の伝送は、次のように実行される。

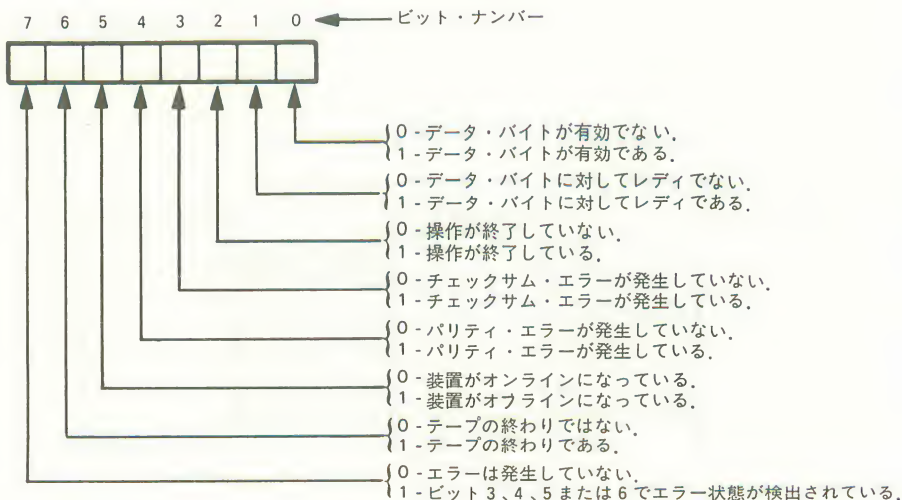
1. システムは、読み込みあるいは書き込みの動作を要求する。
2. システムは、コントローラが1バイトの伝送の用意ができるのを待つ。
3. システムは、コントローラのデータ・ポートとの間で1バイトを伝送する。

4. もし 128 バイトが伝送されれば、テープ・コントローラはブロック全体が伝送されたことを表わすフラグをセットする。伝送が完了していなければ、システムはステップ 2 へ戻る。

テープ・コントローラは、非常に簡単なコマンドの構成で動作する。次に示すコマンド・バイトは、テープ・コントローラによって動作を起こさせるために、テープ・コントローラのコマンド・ポートに送られる。



テープ・コントローラへのコマンドの送付の後、システムはコントローラからステータス・バイトを読み込む。このバイトは次の形式を持っている。



システムがリード動作あるいはライト動作のコマンドを発行すれば、適当なビット（リード動作に対してはビット 0、ライト動作に対してはビット 1）がサンプルされる。テープ・コントローラにおいてデータの送受が可能ならば、システムはテープ・コントローラのデータ・ポートに対して読み込みあるいは書き込み動作を行なう。128回の読み込みまたは書き込みの動作の後に、ビット 2 が 1 となって動作の完了を知らせる。

システムがリワインド動作のコマンドを発行すれば、ステータス・ポートのビット 2 はリワインド動作の完了を表わす。

2.1.1 入 力

テープ・コントローラの特性についての前の記述が与えられれば、人力のパラメータは以下のように指定される。

1. データ・パスの幅. テープ・コントローラからデータは一度に1バイト到着する. システムからの各リード・コマンドは、テープから128バイトのブロックの読み取りを可能とする.
2. データ伝送の速度. この例では、データは同期的に到着する. データ・バイト有効のビットがハイになると、データはCPUの最大速度で入力される.
3. 付随のコントロール情報. この例では、テープ・コントローラはシステムに割り込みを用いていない. システムは、データが有効かどうかを決定するために、テープ・コントローラのステータス・ポートを読み取る.

2.1.2 計算処理

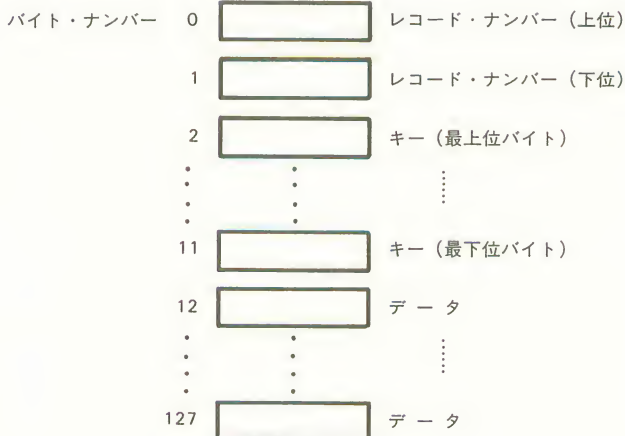
仕様決定のこの部分では、次の要求が考慮される。

- テープから読み出されるデータ・レコードのフォーマット.
- キーがソートされる方法.
- テープに書き込まれるデータ・レコードのフォーマット.

2.1.3 入力レコードのフォーマット

テープから読み出された各データ・レコードは、128バイトから成る. テープから読み出された各データ・レコードには、次の関連した3つのフィールドが存在する.

1. レコード・ナンバー. これは、一意的にレコードを識別する2バイトのフィールドである. レコード・ナンバーは 0000_{16} — $FFFF_{16}$ の範囲にある. レコード・ナンバー $FFFF_{16}$ は E O F (End—Of—File) レコードを指定する.



2. キー. これは10バイトのフィールドである. このフィールドは, レコードの詳細を示すデータを含み, 特定のレコードに対して一意である必要はない. この例では, この10バイトは個人の姓を表わすと仮定している.

3. データ. レコードの残りの 116 バイトはデータを含む.

これらの3つのフィールドは, すべてのレコードに対して17ページの図のように構成されている. 便宜上, データ・レコードのサイズはテープから読み出されるブロックのサイズに等しいことに注意.

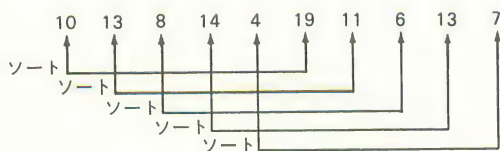
2.1.4 ソートの方法

用いられているソートの方法は, 漸減増加ソート, あるいは Shell ソートである. これは一般に用いられているソートのアルゴリズムであり, “Sorting and Searching,” by D. W. Knuth に詳細に述べられている. 用いられている照合の順序は, A S C I I 照合の順序である. キーは上昇順に分類される.

漸減増加ソートの基本的原理は, 全体のリストが直接挿入を用いてソートされる最終のパスとなるまで, 直接挿入法を用いて順次より大きいサブリストをソートすることである. このソートの利点は, サブリストがソートされるにつれて全体のリストがより整理されたものとなることにある. したがって, 最後のパスで全体のリストがソートされるとき, 必要な交換が少なく, しかも実行時間が減少する. 例として, 10個の要素より成る次のリストを考える.

10 13 8 14 4 19 11 6 13 7

最初のソートのパスでは, 次のようにリストがソートされる.



10 11 6 13 4 19 13 8 14 7

2 番目のパスでは, 次のようにリストがソートされる.



4 7 6 8 10 11 13 13 14 19

そして最後のパスでは全体のリストがソートされる。

基本的なアルゴリズムを次に示す。

与えられるのは、 N レコードである。この場合、レコードは12バイトの長さで、レコード・ナンバーとキー・フィールドから成る。このアルゴリズムには、関連のある2つの変数が存在する。

Increment: 漸減増加ソートでは、サブリスト中の要素数を決定するのに役立つ1組の Increment が選ばれる。この場合、Increment は次のようになる。

$$N/2, N/4, \dots, 1$$

$N/2$, 次に $N/4$, そして最後に 1 (全体のリストがソートされる最後のパスで) を値として含む変数を Increment と呼ぶ。

Subsort counter: Increment の各値に対して、すなわちソートの各パスに対して、この変数は $(N - \text{Increment})$ から N までカウントする。これは各パスにおいて実行されるソートの数を決定する。

アルゴリズムは次のように処理を行なう。

1. Increment = N とする。

Increment = 0 となるまで、ステップ 2 から 12 までを行なう。

2. Increment = Increment / 2

直接挿入ソートを用いて、各サブリストをソートする。

3. Subsort counter = $N - \text{Increment}$

Subsort counter = $N + 1$ となるまで、ステップ 4 から 12 までを行なう。

4. Subsort counter = Subsort counter + 1

5. Keytemp = Key(Subsort counter)

6. Recordtemp = Record(Subsort counter)

7. Index = Subsort counter - Increment

8. Keytemp と Key(Index) を比較。

Keytemp \geq Key(Index) ならばステップ 12 へ、そうでなければステップ 9 へ行く。

9. Record(Index + Increment) = Record(Index)

10. Index = Index - Increment

11. Index > 0 ならばステップ 8 へ、そうでなければステップ 12 へ行く。

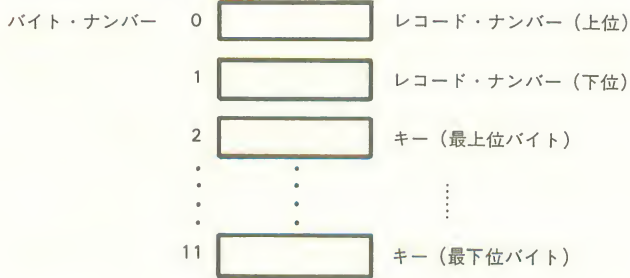
12. Record(Index + Increment) = Recordtemp

2.1.5 出力レコードのフォーマット

テープに書き込まれる各データ・レコードは12バイトから成る。テープに書き込まれる各データ・レコードには関連する次の2つのフィールドがある。

1. レコード・ナンバー。これは、入力レコードのフォーマットにおけるレコード・ナンバーと同一の2バイトのフィールドである。
2. キー。これは、入力レコードのフォーマットにおけるキー・フィールドと同一の10バイトのフィールドである。

これらのレコードは次のように構成されている。



テープのブロックに含まれるバイト数の 128 は、出力レコード中のバイト数である 12 の倍数ではないことに注意。したがって、出力レコードをテープのブロックに詰めるために、何かのアルゴリズムが用いられなければならない。このアルゴリズムは、プログラム設計の項で後述する。

2.1.6 出力

テープ・コントローラの特性についての前述の記述が与えられれば、出力のパラメータは以下のように指定される。

1. データ・パスの幅。データは、テープ・コントローラに一度に 1 バイト送られる。システムからの各書き込み命令によって、テープに 128 バイトのブロックが書き込まれる。
2. データ伝送の速度。この例では、データは同期的に送られる。データ・バイトのレディのビットがハイになれば、データは CPU の最大速度でコントローラに送られる。
3. 付随のコントロール情報。この例では、テープ・コントローラは、データに対するレディを知らせるために、システムに割り込みを用いていない。システムは、テープ・コントローラのデータに対するレディを判断するために、コントローラのステータス・ポートを読み出す。

2.1.7 エラー処理

この例では、関連があるのはテープ・エラーだけである。このエラーは、テープに対する読み込みや書き込みのサブルーチンによって処理される。この処理は第 6 章で論じる。

2.1.8 プログラム設計

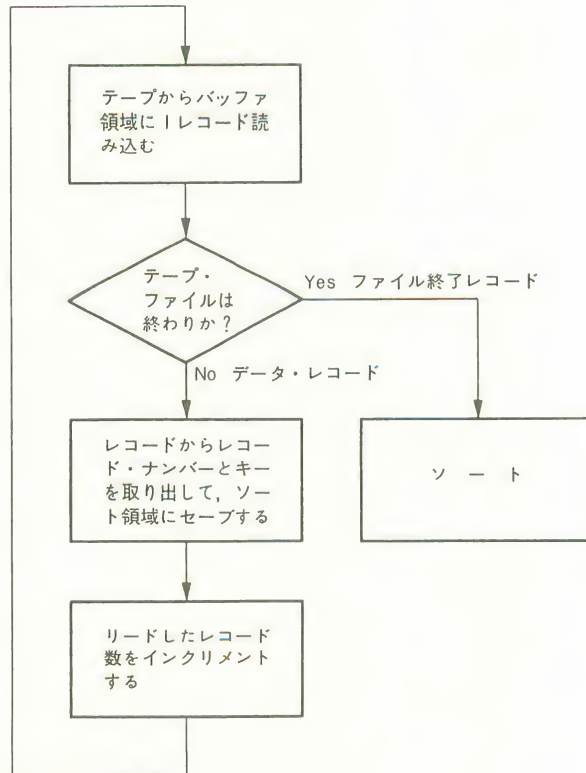
プログラムが行なう作業を検討すると、プログラムを構成する次の 3 つの主要な機能の存在がわかる。

- テープからレコードを読み出し、各レコードからキーを取り出す。
- キーをソートする。
- ソートされたキーを再びテープに書き込む。

前述のモジュールのどれもそれほど複雑ではないので、それらを説明するためにフローチャートを用いる。

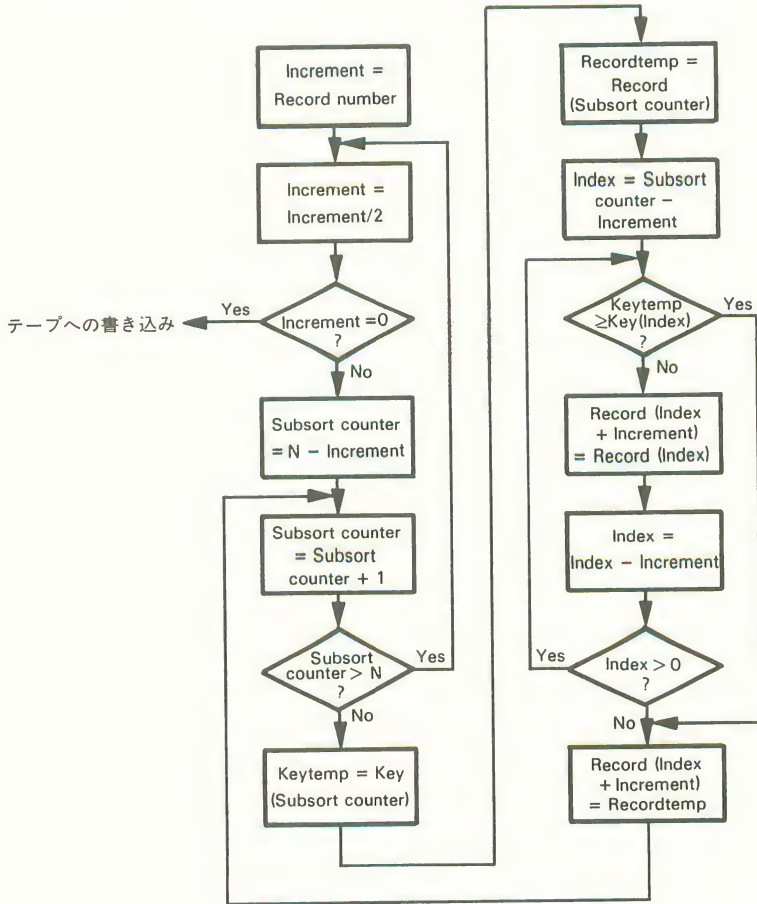
(1) テープからの読み込み

テープを読み込むモジュールには、ただ1つの判断の個所が含まれている。モジュールは、テープからレコードを読み込みながら、そのレコードがEOFレコード（レコード・ナンバー=FFFF₁₆）かどうかを確かめるために、各レコードを調べる。EOFレコードが検出されれば、コントロールはソートのモジュールに渡され、そうでなければ、レコードからレコード・ナンバーとキーが取り出されて、ソートのモジュールによって処理される一時的領域に格納される。それから次のレコードがテープから読み出される。



(2) ソート

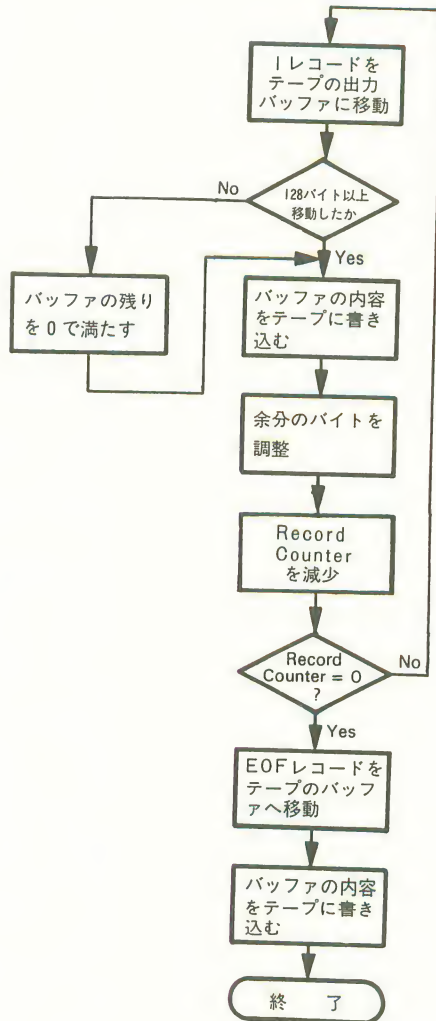
仕様で与えられたソートのアルゴリズムを、このモジュールは実行する。



(3) テープへの書き込み

テープヘキー・ファイルを書き込むモジュールは、テープから読み込みを行なうモジュールのように簡単ではない。モジュールには2つの判断の個所が存在する。第1の判断では、テープ・ブロックを満たさなければならない。各12バイトのレコードに対して128バイトのブロックを書き込むことは、テープ・スペースの点からあまり効率が良くないので、128あるいはそれ以上のバイトが格納されるまで、レコードはバッファ内に編成される。

128バイトが格納されると、判断の時点でバッファはテープに一度に移される。第2の判断の個所には、レコード数の減少が伴う。出力レコードのすべてが移されると、バッファにEOFレコード（レコード・ナンバー＝FFFF₁₆）が付加されてテープに書き込まれる。



第3章

8086アセンブリ言語の 命令セット

8086は、インテルの最初の16ビット・マイクロプロセッサである。1978年に紹介されたとき、従前のマイクロプロセッサと比較して非常に強力なものであった。

8086アセンブリ言語の命令セットは8080 Aと上位方向に互換性がある。ただしソース・プログラム・レベルに限る。すなわち、すべての8080 Aアセンブリ言語の命令は、8086アセンブリ言語の1つあるいは複数の命令に変換できる。だれかが8086アセンブリ言語の命令を、一度に1つずつ、8080 Aアセンブリ言語の1つあるいはそれ以上の命令に変換することを試みる理由はないが、もし行なったならば、すぐさま絶望的にもメモリ配置と特殊な変換規則の矛盾で混乱することになる。これが、8086と8080 Aのアセンブリ言語の命令セットが、“上位方向”に互換性があるという理由である。

8086と8080 Aのアセンブリ言語の命令セットは、オブジェクト・コード・レベルでは互換性はない。このことは、ROMに記憶された8080 Aのプログラムは、8086のシステムでは使用できないことを意味する。

8085と8080 Aのアセンブリ言語の命令セットは、8085のRIMとSIMの命令を除いて、同一である。8085のRIMとSIMの命令は8086の命令に変換できない。これはRIMとSIMの命令が、8086には存在しない8085のシリアルI/Oロジックを用いていることによる。RIMとSIMの命令を除いて、8085と8080 Aのアセンブリ言語の命令セットは同一である。したがって、RIMとSIMの命令は別として、8086アセンブリ言語の命令セットはまた、8085アセンブリ言語の命令セットと上位方向に互換性があることになる。

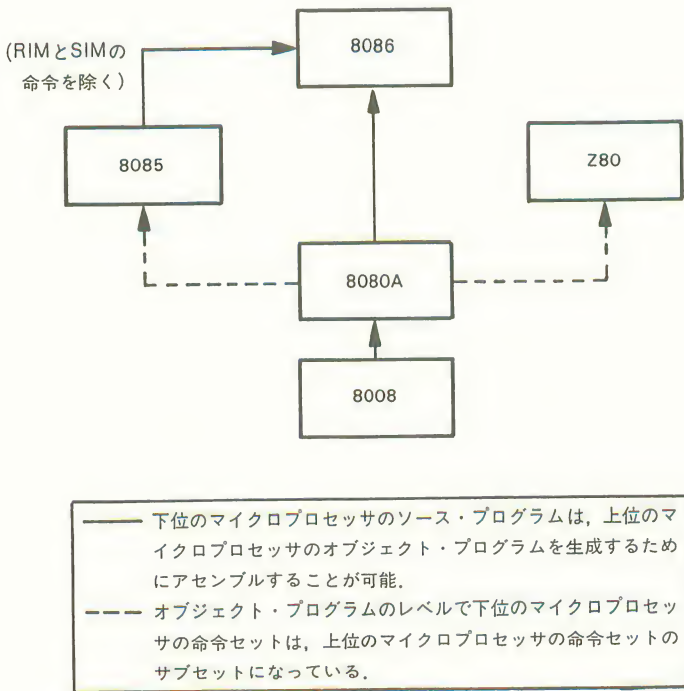
8085と8080 Aのアセンブリ言語の命令セットは、8085のRIMとSIMの命令を除いて、オブジェクト・コードで互換性がある。すなわち、ROMに存在するプログラムは、どちらのマイクロプロセッサでも用いることができる。

8080 Aアセンブリ言語の命令セットは、Z80アセンブリ言語の命令セットのサブセットとなっている。すなわち、Z80は8080 Aのオブジェクト・プログラムを実行できるが、その逆は成り立たない。Z80の全命令セットが用いられているときは、8080 AはZ80のプロ

グラムを実行できない。8086アセンブリ言語の命令セットは、Z80アセンブリ言語の命令セットと上位方向に互換性はない。

歴史的な注釈として、8080Aに先行した8008マイクロプロセッサもまたソース・プログラム・レベルでのみ互換性があったことは述べる価値がある。すなわち、すべての8008アセンブリ言語の命令に対して、8080Aのアセンブリ言語の命令が存在するが、2つのマイクロプロセッサのオブジェクト・コード・セットは同じではない。

以上に述べた種々の命令セットの互換性は、次のように表わせる。

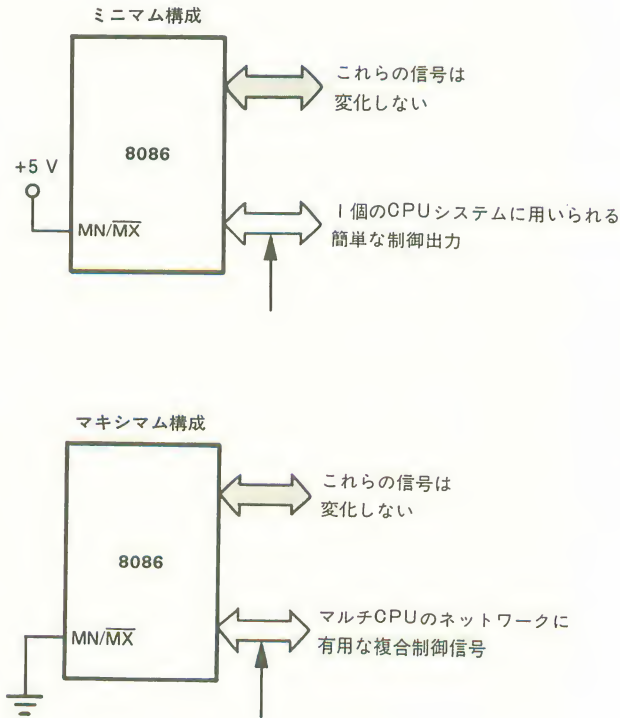


8086のハードウェア設計には、以下のような非常に興味深い革新が見られる。

1. 8086のCPUのロジックは、エグゼキューション・ユニット（EU）とバス・インターフェイス・ユニット（BIU）に分かれている。この両者は非同期に動作する。BIUは、外部バスとのインターフェイスすべてを処理し、外部メモリとI/Oのアドレスの生成を行ない、6バイトの命令オブジェクト・コードのキューを持つ。EUがメモリあるいはI/Oの素子にアクセスする必要があると、BIUに対してバス・アクセスの要求を出す。BIUが現在動作中でなければ、EUからのバス・アクセスの

要求を受け付ける。E Uからの有効な保留されたバス・アクセスの要求がなければ、B I Uは6バイトの命令オブジェクト・コードのキューを満たすために、命令フェッチのマシン・サイクルを実行する。C P Uはキューの前から命令オブジェクト・コードを得る。したがって、命令フェッチの時間は大きく削除される。

2. 8086は、簡単な1個のC P Uシステムから複数のC P Uのネットワークまで、広い範囲のマイクロコンピュータ・システム構成で動作するように設計されている。この大きい融通性をサポートするために、8086ピンの出力のいくつかは信号の切り換えを行なう。これは以下のように説明できる。



MN/\overline{MX} のレベルに基づいて、同一のピンがこれら2つの信号の組を出力する。この大規模な信号の再配置は、マイクロプロセッサ産業にとって最初は非常に空想的であり革新的なものであった。

3. 8086は、マルチC P U構成でバス・アクセスのプライオリティを処理するためのロジックが組み込まれている（これは新しい概念ではなく、National SemiconductorのS C /M Pでは何年も用いられている）。

4. マルチCPU構成では、各々の8086CPUはそれ自身のローカル・メモリを持つことが可能であり、同時に共通メモリを共有する。共通メモリはすべてのCPUあるいは特定のCPUによって共有される。
5. 8086は、ミニコンピュータの領域であるプログラムが強調される応用分野においても、有効に競えるように設計されている。外部メモリの1メガバイトまで、直接にアドレス指定ができる。すべてのメモリのアドレス指定はベース相対となっている。このメモリ・アドレス指定方法は当然、リロケイト（再配置）可能なオブジェクト・プログラムを生成する（リロケイト可能なオブジェクト・プログラムは、1つのメモリ・アドレス域から他へ移動して修正なしに再び実行することができる）。また、8086はスタック相対のアドレス指定を用いているので、リエントラント・プログラムが容易に書ける（リエントラント・プログラムは、実行の途中で中断して再び実行することができる。たとえば、自分自身を呼び出すサブルーチンはリエントラントであり、外部割り込みによって実行の途中で中断されて、インタラプト・サービス・ルーチン内で再び実行できるプログラムもまたリエントラントである）。
6. 8086は、後続の命令のオブジェクト・コードの解釈を修飾するプレフィックス命令を用いている。

8086は、8080Aのように、実際は複数チップのマイクロプロセッサ構成の一要素である。

8086マイクロプロセッサ自身に加えて、8284クロック・ジェネレータ/ドライバが必要である。他のロジックを用いて必要なクロック信号を作ることできるが、これは実用的でも経済的でもない。

8086マイクロプロセッサ構成で必要な3番目の素子は、8288バス・コントローラである。

8080Aとシステム・バスの間には、通常8228システム・バス・コントローラを用いるように、8086とシステム・バス（複数の場合もある）の間には普通8288バス・コントローラを用いる。しかし8086の場合、シングル・バス構成では、何の不利も受けずに、8288バス・コントローラを省略することができる。

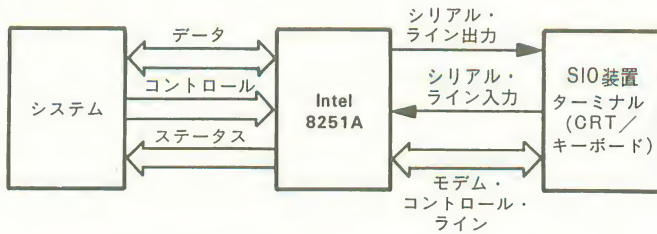
6, 7, 8, と9章では、基本的な8086のハードウェア、シングルCPU構成、Multibus*, さらにマルチCPU構成について論じる。

3.1 I/Oドライバ

次に、システムとシリアルの入出力チャンネルとのインターフェイスを行なうプログラム・モジュールの詳細を述べる。

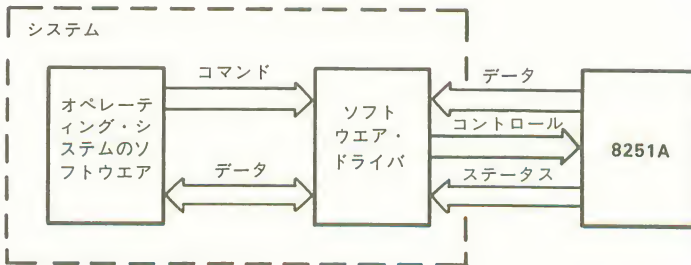
以下にシリアルな入出力チャンネルの一般的なブロック図を示す。

* Multibus は Intel Corporation の登録商標である。



この例では、シリアル入出力 (SIO) のチャンネルは、インテルの8251A プログラム・コミュニケーション・インターフェイスである。8251Aは通信端末に接続されていると仮定する。通信端末は、システムから伝送されたデータを表示するCRTを有している。キーボードからのデータは、チャンネルによってシステムへ伝送される。入力あるいは出力に端末で、データにバッファは用いられない。データの送信と受信は非同期に行なわれる。

ドライバとも呼ばれるプログラム・モジュールは、オペレーティング・システムのソフトウェアと8251Aとを結合している。これは以下のように説明できる。



オペレーティング・システムのソフトウェアは、I/O ドライバのプログラム・モジュールにコマンドやデータを送る。これは種々の方法で処理される。それを以下に示す。

1. コマンドやデータをレジスタに設定する。たとえば、コマンド保持に1つのレジスタを割り当て、データは他のレジスタを通して受け渡す。
2. タスク・ブロックを用いる。タスク・ブロックには、コマンドとデータあるいはコマンドとデータのポインタを含むことができる。タスク・ブロックは、固有のメモリ位置に配置するか、あるいはレジスタの1つで位置を表わすことができる。
3. スタックを用いる。システム・ソフトウェアは、タスク・ブロックと同等のもの（すなわち、コマンドとデータあるいはポインタ）をスタックにプッシュすることができる。

上記方法からの選択は、一般にプロセッサに依存して決定される。現在の議論はプロセッサに依存していないので、パラメータ受け渡しの方法を選択するための原理は、後の章に譲る。

3.1.1 入 力

この例で用いられているSIO素子はインテルの8251Aである。この素子は以下の入力仕様を必要とする。

1. データ・パスの幅。8251Aでは、5, 6, 7, あるいは8ビットのキャラクタが許されている。この例では、コマンドとステータスに対して、素子で8ビットのデータ・パスを必要とする。異なる大きさのデータ・パスを用いる将来のシステムを可能とするために、プログラム設計ではデータ・パスの大きさを指定することができる。
2. データ転送速度。この例では、データは非同期に転送される。最大データ転送速度だけが指定できる。この例では、最大データ転送速度として、9600ボー（baud）が指定されている。
3. ハンドシェイクのプロトコル。この例では、SIOチャンネルはマイクロプロセッサにインタラプトを用いているのではなく、マイクロプロセッサが、データが有効かどうかを決めるために、チャンネルを調べる。

次に、I/Oドライバは実際のI/Oチャンネルの動作を考慮する必要がある。データの転送に加えて、8251Aの場合には、I/Oチャンネルにコントロール情報を送信し、それからステータスを受信しなければならない。

データ・ポートは次のような構成となる。

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
-----	-----	-----	-----	-----	-----	-----	-----

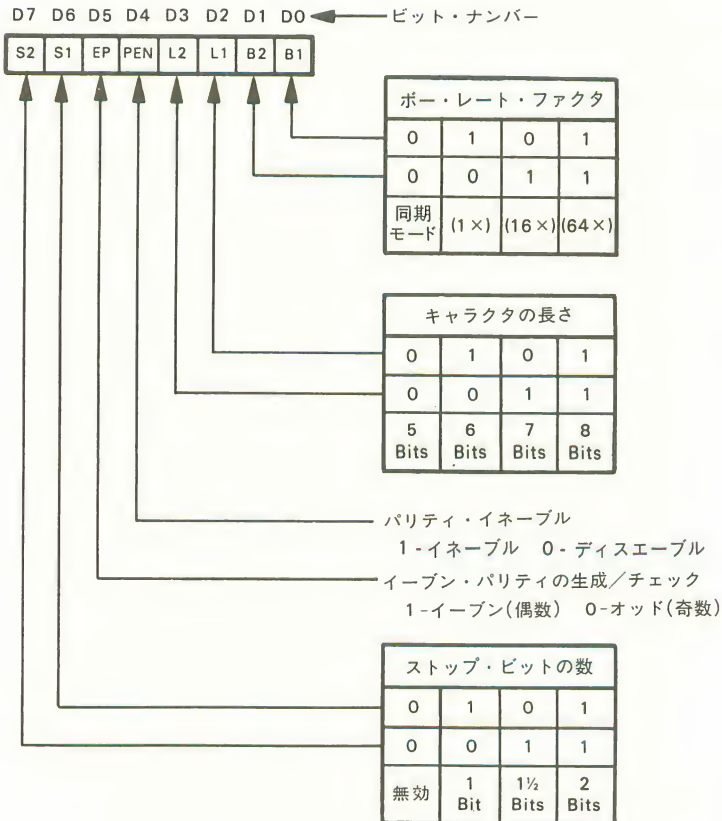
適当なステータス・ビットが1に設定されると、このI/Oポートのデータが有効であることが知られている。8251Aの場合、ステータス・ポートのRxRDY（レシーバ・レディ）ビットが1になる必要がある。8251Aは、コントロール・ポートに情報を書き込むことによって、既知の状態に初期化される。8251Aの初期化には、少なくとも2バイトのコントロール情報を必要とする。コントロール情報は次の順序で送られる。

1. モード・セレクト・バイト
2. Sync キャラクタ 1（同期モードのみ）
3. Sync キャラクタ 2（同期モードのみ）
4. コマンド・セレクト・バイト

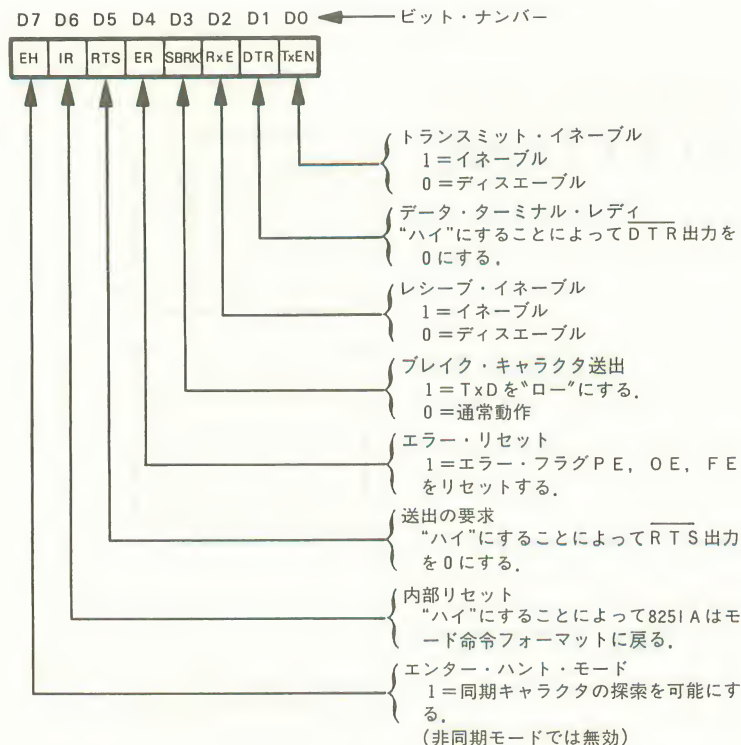
この例では、8251Aは非同期モードで動作させるので、以下の初期化の2バイトが必要となる。

1. モード・セレクト・バイト
2. コマンド・セレクト・バイト

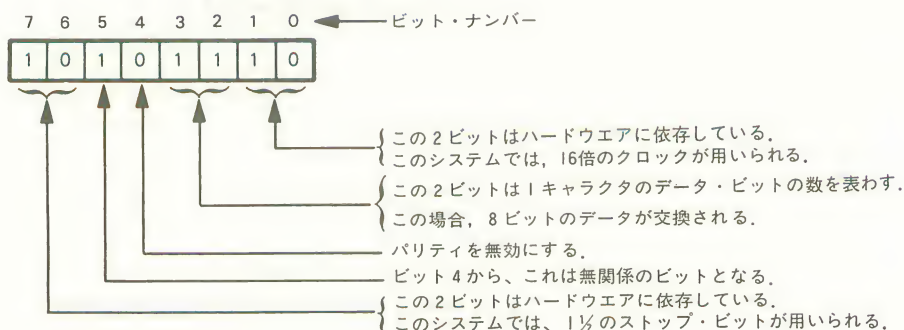
モード・セレクト・バイトの形式を次に示す。



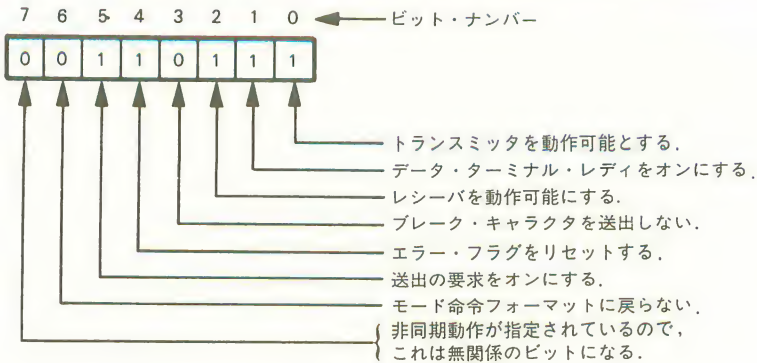
コマンド・セレクト・バイトの形式を次に示す。



前述の仕様から、モード・セレクト・バイトは次のようになる。



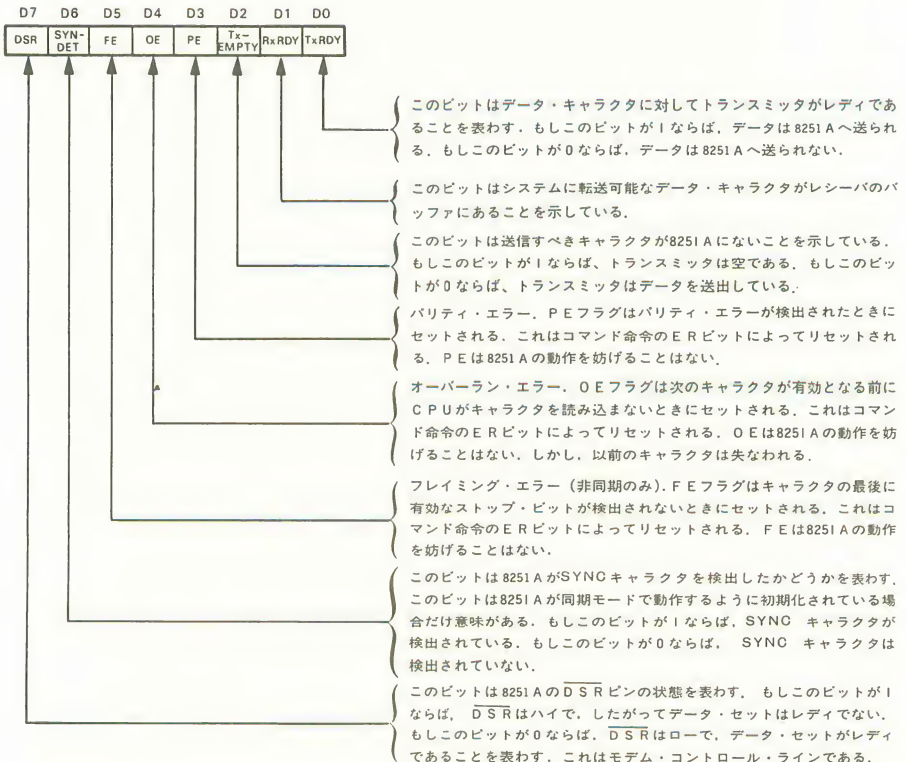
最初のコマンド・セレクト・バイトは次のようになる。



8251A プログラミングのその他の特徴は、第6章で解説する。

ステータス・ポートは、8251Aの状態と、それに接続されている装置の状態についての情報を与える。

ステータス・ポートから1バイトが読み出されると、以下の情報がシステムに転送される。



3.1.2 計算処理

どのような種類の機能をドライバは備えているべきか。また入力や出力でデータはどのように変換されるのか。以下に実際のI/Oドライバが備えている機能を示す。

1. チャンネルの初期化：電源がシステムに供給されると、8251Aは不明の状態であち上がる。I/Oドライバはチャンネルを既知の状態に設定する。
2. 1個のキャラクタの入力：この機能が要求されると、ドライバは、ステータス・ポートから読み出しを行ない、データが有効になるまで待つ。データが有効になれば、ドライバはデータ・ポートを読み出してシステムに情報を受け渡す。
3. 1個のキャラクタの出力：この機能が要求されると、システムは、出力されるべきキャラクタあるいはそのキャラクタのポインタを、ドライバに受け渡さなければならない。ドライバは、ステータス・ポートの読み出しを行ない、トランスミッタが利用可能となるまで待つ。トランスミッタが利用可能となれば、ドライバは指定されたキャラクタをデータ・ポートに転送する。
4. チャンネル状態のチェック：場合によっては、システムはキャラクタを読み出す必要がなく、むしろキャラクタが有効かどうかを知る必要がある。このような状況では、システムはステータス・ポートの内容を読み出す。
5. コントロール情報のチャンネルへの送出：システムは、たとえばチャンネルがパリティ・エラーのチェックをできるように、チャンネルの状態を変える必要がある。
6. チャンネルからの一連のキャラクタの入力：ある終結条件が検出されるまで、キャラクタの入力を必要とする場合がある。たとえば、キャリッジ・リターンが終結条件を構成するか、あるいは一定数のキャラクタが入力される。たとえば、5つの数字がZIPコード*を構成する。I/Oドライバはチャンネルからデータを読み出す。これには、データが有効になるのを待ち、データ・ポートの情報を読み出してメモリの指定された場所にデータをセーブして、終結条件に達したかを決定するために調べることが含まれる。
7. チャンネルへの一連のキャラクタの出力：システムは、終結条件が検出されるまで、一連のキャラクタの出力を要求する。終結条件には、前もって決められたストリング最後のキャラクタの検出あるいは特定数のキャラクタの出力が含まれる。I/Oドライバは終結条件を調べる。終結条件が検出されなければ、I/Oドライバは、特定のメモリ位置からデータをロードして、チャンネルにデータを送出する。

3.1.3 出力

8251Aは、チャンネルの出力特性を定義するために、コントロール情報を用いる。定義される必要のある出力仕様を以下に示す。

1. データ・パスの幅：8251Aは、5、6、7、あるいは8ビットよりなるデータ単位

* 日本では郵便番号（訳者注）

が可能である。このシステムでは、8データ・ビットが転送される。

2. データ転送速度：この場合、最大のデータ速度は9600ボーとなる。

3. ハンドシェイクのプロトコル：この例では、8251Aがシステムに割り込みを用いるのではなく、システムは、8251Aが他のデータを送信できるかを判断するために、8251Aのステータス・レジスタを読み出す。

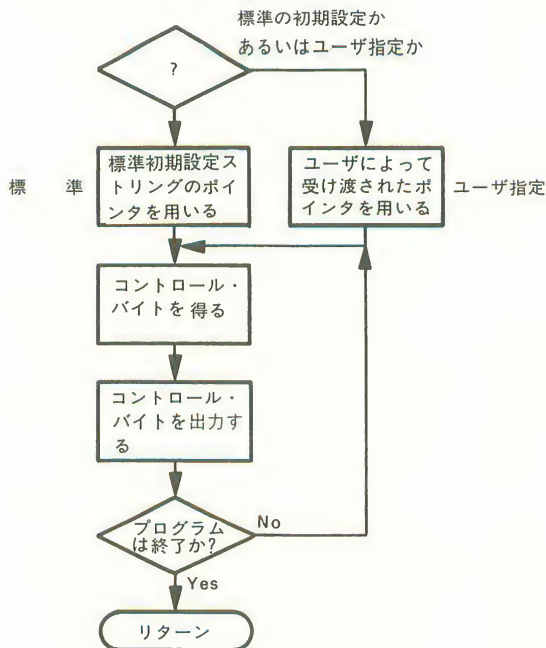
8251Aのデータとコントロール/ステータスのポートについては既に述べた。ステータス・レジスタのTxRDY（トランスミッタ・レディ）ビットが1になると、データがチャンネルに送られる。

3.1.4 プログラム設計

この項では、特定のモジュールで複雑なものはない。このことから、I/Oドライバの各々のモジュールについて述べるのに、フローチャートを用いる。

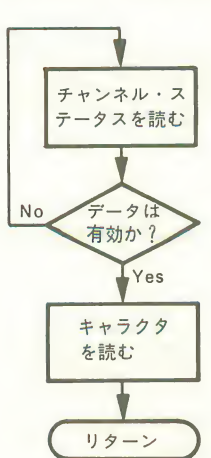
(1) 初期設定

初期設定ルーチンには1つだけ、主要な決定項目が含まれている。標準的な初期設定が必要ならば、標準初期設定手順に対するポインタが、コントロール・ポートへ送られるべき情報を識別する。他の手段として、“カスタム”の初期設定手順が実行できる。この場合、ユーザは初期設定手順とそれに対するポインタを用意しなければならない。

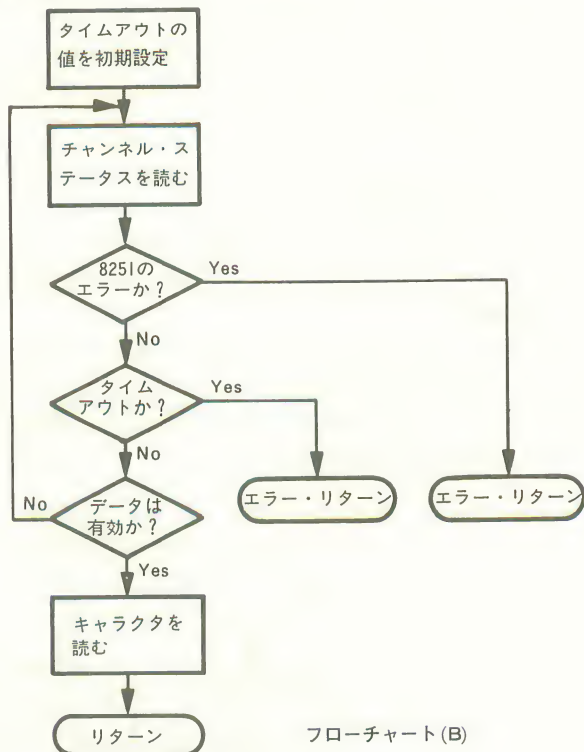


(2) 1個のキャラクタの入力

単一キャラクタ入力ルーチンのフローチャートは (A) に示されている。チャンネル・ステータスを読み出すルーチン呼び出して、データが有効になるのを待ち、データ・ポートからデータを読み出して復帰する。



フローチャート (A)



フローチャート (B)

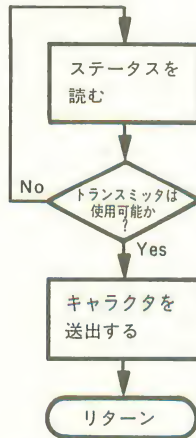
単一キャラクタ入力ルーチンの設計には、次の2つの大事な点が含まれていない。

- 8251Aのエラー処理。チャンネル・ステータスを読み出すルーチンが呼び出されると、8251Aのステータス・レジスタのエラー・ビットが調べられる。もし8251Aのエラーが検出されれば、単一キャラクタ入力ルーチンによってI/Oドライバに適切なエラー・コードが返される。
- タイムアウト・エラー。ドライバは、タイムアウト・クロックとして用いるために、適切なレジスタあるいはメモリ位置の初期設定を行ない、チャンネル・ステータスの読み出しを行なうルーチン呼び出すたびに、その内容を減じる。タイムアウトのレジスタあるいはメモリ位置の内容が減少して0になれば、タイムアウトのエラー・コードを呼び出し元ルーチンへ返す。

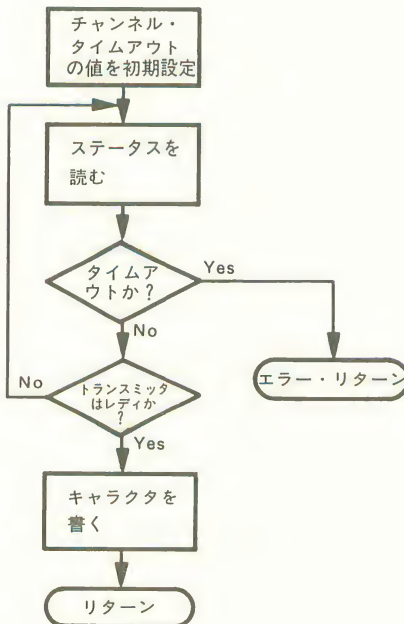
これらの考慮を単一キャラクタ入力ルーチンに加えると、フローチャートは (B) のように修正される。

(3) 1個のキャラクタの出力

単一キャラクタ出力ルーチンのフローチャートを以下に示す。チャンネル・ステータスを読み出すルーチンを呼び出して、トランスミッタが使用可能になるのを待ち、データ・ポートにキャラクタを書き込んで復帰する。



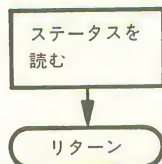
入力ルーチンと同じく、上に示したように、最初の設計にはエラーとタイムアウトについての考慮は含まれていない。8251Aの場合、ステータス・レジスタでは送信エラーが通知されないで、調べる必要のあるエラー状態は存在しない。しかし、タイムアウトのチェックは含まれるので、プログラムのフローチャートは以下のように修正される。



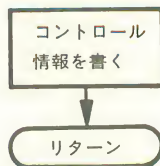
(4) チャンネル・ステータスのチェックとコントロール情報の送出

チャンネル・ステータスを読み出すルーチンとコントロール情報を送出するルーチンは、それぞれ簡単なフローチャートとなる。それらを以下に示す。

チャンネル・ステータスのチェック

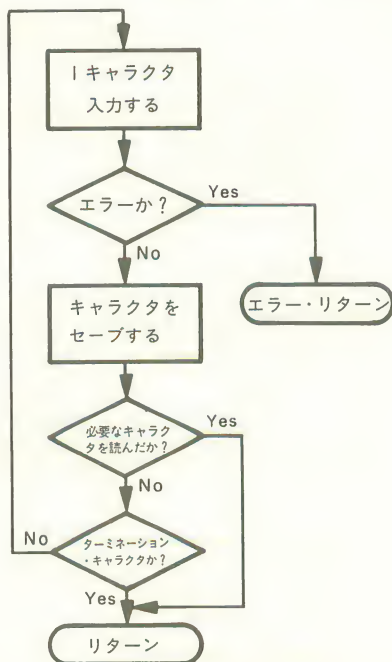


コントロール情報の送出



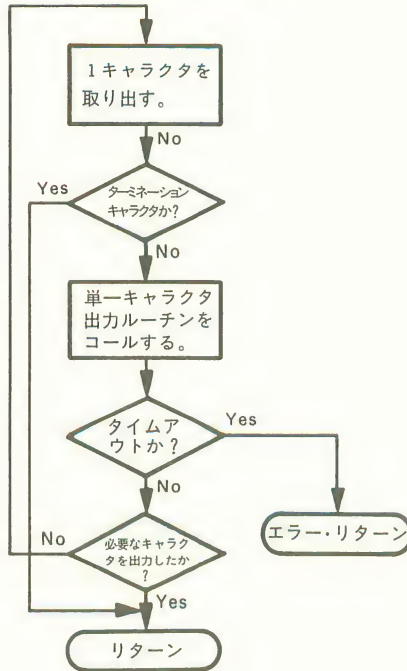
(5) 一連のキャラクタの入力

複数キャラクタの入力ルーチンは、データを読み込むために、単一キャラクタ入力ルーチンを用いている。単一キャラクタ入力ルーチンから復帰すると、エラーのチェックが行なわれる。単一キャラクタ入力ルーチンでエラーが検出されると、このエラーは呼び出し元ルーチンに受け渡される。呼び出し元プログラムで指定された位置にキャラクタのセーブを行なった後、複数キャラクタ入力ルーチンは終結条件を調べる。ルーチンで、指定された数のキャラクタが読み込まれたか、ターミネーション・キャラクタが読み込まれていれば、複数キャラクタ入力ルーチンは呼び出し元プログラムに復帰する。



(6) 一連のキャラクタの出力

複数キャラクタ出力ルーチンは、ユーザ指定の位置からキャラクタをロードする。キャラクタがターミネーション・キャラクタであれば、複数キャラクタ出力ルーチンは呼び出し元プログラムに復帰する。そうでなければ、キャラクタは単一キャラクタ出力ルーチンに送られる。単一キャラクタ出力ルーチンから復帰すると、タイムアウトのチェックが行なわれる。タイムアウトが検出されれば、それは呼び出し元プログラムに受け渡される。複数キャラクタ出力ルーチンは、次に指定された数のキャラクタを出力したかチェックする。もししていれば、呼び出し元プログラムに復帰する。



3.2 8086の命令セット

8086の命令セットは、16ビット・マイクロプロセッサの新しい世代を代表する複雑なものとなっている。8086の命令セットは約70の基本命令より成り、メモリ参照命令では30にも及ぶアドレッシング・モードが利用できる。

CPUの命令セットの記述には、以下の基本的な情報が含まれる必要がある。

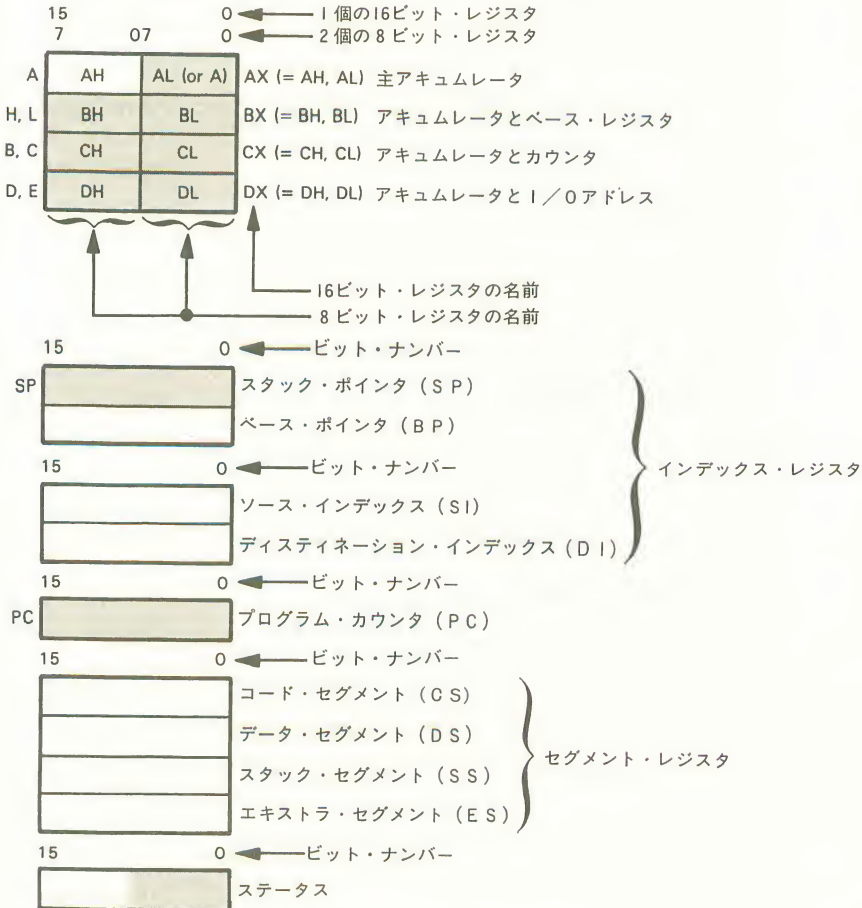
1. CPUの構成はどのようになっているのか。すなわち、どのようなレジスタとステータスが利用できるのか。各レジスタの主要な用途は何か。
2. どのような命令が利用できるのか。明らかに、各命令の機能に関連した解説と共に、命令セットの内容豊富な一覧表が必要である。この一覧表は、いくつかある方法の1

つで構成される。この章では、個々の命令を見つけやすいように、命令をアルファベット順にあげてある。次章では、命令をバイトあるいはグループで調べられるような機能（たとえば、算術演算操作は1つの節で論じている）に従って命令をあげてある。

3. CPUはどのような型のデータを処理するのか。簡単なCPUでは、すべてのデータは1つの形式で、多分バイトとして、処理されることを必要としている。より融通性のあるCPUでは、個々のビット、バイト、16ビットのワード、32ビット長のワードとして、データのアドレス指定を選択できる。
 4. オペランドのソースとディスティネーションにどのようなアドレス指定が選べるのか。簡単なマイクロプロセッサでは、メモリとCPUのレジスタとの間でデータの移動を行なう命令によってのみ、メモリのアドレス指定が可能であり、データについてのすべての操作は、オペランドがCPUのレジスタに存在することを必要とする。より複雑なマイクロプロセッサでは、1つのオペランドはメモリからフェッチすることが可能で、もう一方のオペランドはCPUのレジスタに存在する。ある場合には、両方のオペランドがメモリに存在できるCPUもある。メモリ・オペランドの重要性を判断するときは、可能なメモリ・アドレス指定の選択を評価する必要がある。
 5. どの命令にどのようなアドレッシング・モードが可能か。特定の命令に対してどのようなアドレッシング・モードが可能かを知ることが、命令セットの有効な利用へのかぎとなる。しかし、各命令に対して可能なすべてのアドレッシング・モードを記述しようとしたならば、この本は15巻のセットにもなってしまうだろう。したがって、命令セットの一覧の前に、アドレッシング・モードの節を設けている。
 6. 命令の多数のグループは、CPUのステータス・レジスタにどのように影響するのか。アセンブリ言語において各種の条件の表現を評価するためには、条件がアセンブリ言語にどのように変換されるかを知る必要がある。命令がステータス・フラグにどのように影響するかを知ることによって、プログラムはアセンブリ言語で条件付き表現を書くことが可能となる。
 7. ある場合には、他の情報が重要となる。たとえば、特定の命令が実行に要するサイクル数、あるいは命令が占めるプログラム・メモリのバイト数である。この場合、各命令の記述には実行に要するサイクル数が示されている。しかし、各命令が必要とするバイト数は、場合によって指定されるアドレッシング・モードに大きく依存している。
- 8086の命令セットのここでの議論は、以下の順序で進められている。
1. 8086のレジスタとステータス・レジスタについて、命令の各グループによってステータスがどのように影響を受けるかについての解説。ステータス・レジスタはまた、フラグ・レジスタあるいはプログラム・ステータス・ワードと呼ばれる。
 2. 8086のアドレッシング・モードについての解説。
 3. 8086の各命令についての解説。このセクションの前には、命令記述に用いられるシンボルと用語の要約が示されている。

3.3 8086のレジスタとフラグ

8086には、4個の16ビット汎用レジスタ、2個の16ビット・ポインタ・レジスタ、2個の16ビット・インデックス・レジスタ、1個の16ビット・プログラム・カウンタ、4個の16ビット・セグメント・レジスタ、さらに1個の16ビット・フラグ・レジスタがある。これらのレジスタを以下に示す。

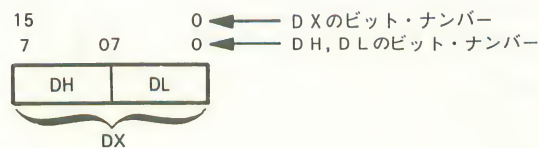
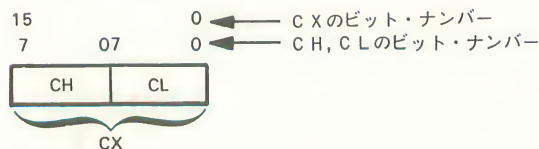
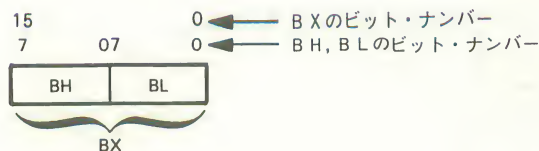
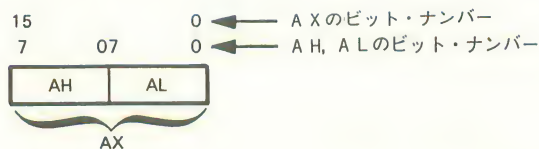


8080 A レジスタに相当するもの。

8080 A レジスタの名前は左側余白に示されている。

3.3.1 汎用レジスタ

汎用レジスタは、2つの別々の8ビット・レジスタとして参照される。それを以下に示す。



これは、8ビット数値に、8ビット操作よりも時間とメモリ領域を必要とする16ビット操作を行なう代わりに、8ビット操作を行なえることで有利となる。たとえば、レジスタに200₁₀を初期設定して、続く事象に基づいて0に減少させる場合には、8ビットのレジスタで明らかに十分である。

汎用レジスタは、すべての8あるいは16ビットの算術演算や論理演算のオペランドとして用いることができる。

AXレジスタは主要なアキュムレータとして用いられる。このレジスタは他にはない2つの特徴を持つ。すべてのI/O操作はこのレジスタを通して行なわれ、イミディエイト・データを用いる操作が、このレジスタに対して行なわれたときは、普通必要なメモリ領域は少なくなる。さらに、ストリング操作と算術演算の命令のあるものはこのレジスタを用いる。

ALレジスタは一般に、8080AのAレジスタに対応している。

BXレジスタはベース・レジスタと呼ばれる。これは、8086のメモリ・アドレスの計算に用いられる唯一の汎用レジスタである。メモリ・アドレスの計算にこのレジスタを用いるメモリ参照はすべて、デフォルト・セグメント・レジスタとしてDSレジスタを用いる。BXレジスタは一般に、8080AのHLレジスタに対応している。すなわち、BHレジスタは8080AのHレジスタに、BLレジスタは8080AのLレジスタに対応する。

CXレジスタはカウント・レジスタと呼ばれる。このレジスタは、ストリングとループの操作で減少する。CXは普通ループの繰返し回数を制御するために用いられる。また、複数ビットのシフトとローテートにも用いられる。このレジスタは一般に、8080AのBCレジスタに対応している。

CHレジスタは8080AのBレジスタに対応している。CLレジスタは8080AのCレジスタに対応している。

DXレジスタは、ほとんどニーモニックの理由から、データ・レジスタと呼ばれる。このレジスタはI/O 命令に対してI/O のアドレスを与える。この機能は8086の他のレジスタでは行なえない。このレジスタは一般に、8080AのDEレジスタに対応している。

DHレジスタは8080AのDレジスタに対応し、DLレジスタは8080AのEレジスタに対応する。DXレジスタはまた、乗算と除算を含む算術演算操作にも用いられる。

3.3.2 ポインタ・レジスタ

ポインタ・レジスタは、スタック・セグメント内のデータにアクセスするために用いられる。それらはすべて16ビットの算術演算あるいは論理演算の操作で、オペランドとして用いられる。

SPレジスタは、スタック・ポインタと呼ばれ、メモリのスタックを可能にする。メモリ・アドレス指定にSPを参照するものはすべて、セグメント・レジスタとしてSSレジスタを用いる。このレジスタは一般に、8080AのSPレジスタに対応している。

BPレジスタは、ベース・ポインタと呼ばれ、スタック・セグメント中のデータをアクセス可能にする。通常は、スタックを通して受け渡されるパラメータを参照するためにこのレジスタが用いられる。

3.3.3 インデックス・レジスタ

インデックス・レジスタは、データ・メモリ内のデータにアクセスするために用いられる。インデックス・レジスタは、ストリング操作で広く用いられる。すべての16ビットの算術演算や論理演算の操作では、オペランドとして用いられている。

3.3.4 セグメント・レジスタ

セグメント・レジスタは、すべてのメモリ・アドレス指定の計算に含まれている。各セグメント・レジスタは、8086のメモリ・アドレス領域で、そのセグメント・レジスタのカレント・セグメントと呼ばれる、64キロのメモリのブロックを定義する。たとえば、DSレジスタは、カレント・データ・セグメントと呼ばれる64キロのセグメントを定義する。

CSレジスタは、コード・セグメント・レジスタとしても知られている。各命令のフェッチの間に、フェッチされるべき命令のメモリ・アドレスの計算のために、プログラム・カウンタの内容はCSレジスタの内容に加算される。

DSレジスタは、データ・セグメント・レジスタとしても知られている。すべてのデータ・メモリの参照は、次の3つの例外を除いて、データ・セグメント・レジスタ相対となる。

1. スタック・アドレスは、スタック・ポインタを用いて計算される。
2. BPレジスタを用いて計算されるデータ・メモリ・アドレスは、スタック・セグメント相対となる。
3. ストリング操作（アドレス計算にDIレジスタを用いるもの）は、エキストラ・セグメント相対となる。

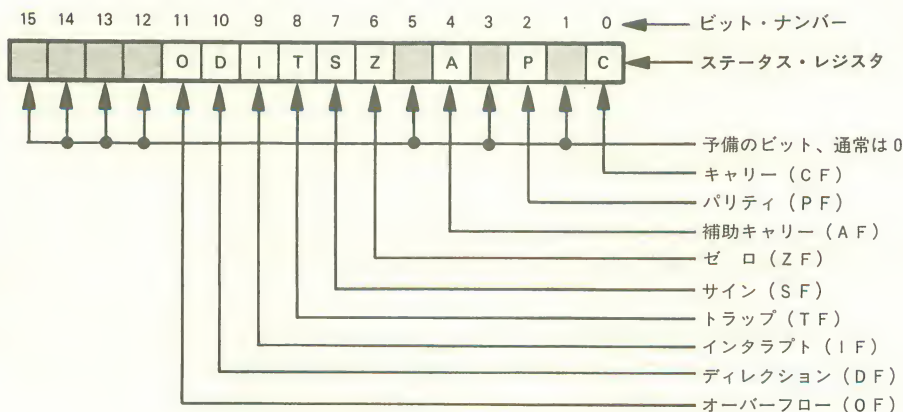
SSレジスタは、スタック・セグメント・レジスタとも呼ばれる。アドレス計算にSPあるいはBPのレジスタを用いるデータ・メモリの参照はすべて、SSレジスタ相対となる。したがって、スタックに対する命令（たとえば、PUSH, POP, CALL, RETやINTなど）は、セグメント・レジスタとしてSSレジスタを用いる。

ESレジスタは、エキストラ・セグメント・レジスタとも呼ばれる。ストリング操作は、ESレジスタ相対となり、DIレジスタを用いてメモリ・アドレスを計算する。

セグメント・レジスタの使用は、一般に命令によって暗黙に指定されているが、ほとんどの場合にこのセグメント・レジスタの変更を可能とする機構については後で論じる。

3.3.5 フラグ・レジスタ

8086は、ステータス・レジスタあるいはプログラム・ステータス・ワードとも呼ばれる16ビットのフラグ・レジスタを持つ。このレジスタを以下に示す。



キャリー，補助キャリー，オーバーフロー，サインのフラグは全く標準的なものである。

キャリー・フラグは，算術演算による上位ビットからのキャリーを表わす。キャリーはまたシフトやローテートの命令によっても変化する。

オーバーフロー・フラグは，算術演算による最上位ビットへのキャリーと最上位ビットからのキャリーのエクスクルーシブORをとったものである*。これは，符号付き2進演算のオーバーフローを意味する。

サイン・フラグは，算術演算操作による最上位ビットに等しい。符号付き2進演算が行なわれていると仮定すれば，サイン・フラグの0は正の結果を表わし，サイン・フラグの1は負の結果を表わす。

補助キャリーは，8080Aの同じ名前のフラグと同一である。これは，8ビットのデータ単位で，ビット3からのキャリーを表わす。

減算命令では，減数から被減数を減算するために2の補数演算を用いる。ただし，キャリー・フラグは反転している。すなわち，減算操作によって，最上位ビットからのキャリーがなければキャリー・フラグは1にセットされ，最上位ビットからのキャリーがあれば0にリセットされる。したがって，キャリー・フラグは Borrow を表わす。

パリティ・フラグは，データ操作の結果の下位8ビットに偶数個の1のビットがあれば1にセットされる。1のビットが奇数個あれば，パリティ・フラグは0にリセットされる。

ゼロ・フラグは，データ操作の結果が0ならば1にセットされ，データ操作の結果が0でなければ0にリセットされる。

ディレクション・フラグは，ストリング操作でインデックス・レジスタの内容を自動増加させるかあるいは自動減少させるかを定める。ディレクション・フラグが1ならば，SIとDIのインデックス・レジスタの内容は減少する。すなわち，ストリングは最上位のメモリ・アドレスから最下位のメモリ・アドレスへとアクセスが行なわれる。ディレクション・フラグが0ならば，SIとDIのインデックス・レジスタの内容は増加する。すなわち，ストリングは最下位のメモリ・アドレスから始まってアクセスされる。

インタラプト・フラグは，割り込みの有効・無効を支配する。このフラグは，8086の割り込みを有効とするためには1でなければならない。このフラグが0ならば，すべての割り込みは無効となる。

トラップ・フラグは，8086を“シングル・ステップ”のモードにする特殊なデバッグ用の助けとなる。シングル・ステップのモードは，その存在が割り込みのロジックに依存するので，8086の割り込みのロジックと共に詳細を述べる。

キャリー，補助キャリー，パリティ，サイン，さらにゼロのフラグは，8080Aにも見られる。オーバーフロー，ディレクション，インタラプト，トラップのフラグは8086に新しいものである。

* この結果が1のときだけ，符号付き2進演算の加減算でオーバーフローが生じている（訳者注）。

3.3.6 命令がどのようにフラグ・レジスタに作用するか

以下の一覧は、個々の命令とそれがフラグ・レジスタに与える影響について示した表を識別する。たとえば、ADD命令のフラグへの影響については、表3-2を参照。

命令のニーモニック	表	命令のニーモニック	表
AAA	3-4	LODS	3-1
AAD	3-10	LOOP 命令	3-1
AAM	3-10	MOV	3-1
AAS	3-4	MOVS	3-1
ADC	3-2	MUL	3-6
ADD	3-2	NEG	3-2
AND	3-7	NOP	3-1
CALL	3-1	NOT	3-1
CBW	3-1	OR	3-7
CLC	3-9	OUT	3-1
CLD	3-9	POP	3-1
CLI	3-9	POPF	3-12
CMC	3-9	PUSH	3-1
CMP	3-2	PUSHF	3-1
CMPS	3-2	RCL	3-8
CWD	3-1	RCR	3-8
DAA	3-5	REP	3-1
DAS	3-5	RET	3-1
DEC	3-3	ROL	3-8
DIV	3-11	ROR	3-8
ESC	3-1	SAHF	3-9
HLT	3-1	SAR	3-7
IDIV	3-11	SBB	3-2
IMUL	3-6	SCAS	3-2
IN	3-1	SEG	3-1
INC	3-3	SHL	3-7
INT	3-13	SHR	3-7
INTO	3-13	STC	3-9
IRET	3-12	STD	3-9
Jump-on-Conditions (条件付分岐)	3-1	STI	3-9
JCXZ	3-1	STOS	3-1
JMP	3-1	SUB	3-2
LAHF	3-1	TEST	3-7
LDS	3-1	WAIT	3-1
LEA	3-1	XCHG	3-1
LES	3-1	XLAT	3-1
LOCK	3-1	XOR	3-7

(1) 何の影響も与えない

表3-1の命令は、8086のどのフラグにも何の影響も与えない。

(2) すべての算術演算のフラグに影響を与える

表3-2の命令は、8086の6つの算術演算のフラグ、オーバーフロー、キャリー、補助キャリー、ゼロ、サイン、パリティのすべてに影響を与える。

(3) キャリーを除くすべての算術演算のフラグに影響を与える

表3-3の命令は、8086のキャリーを除くすべての算術演算のフラグに影響を与える。オーバーフロー、補助キャリー、ゼロ、サイン、パリティはすべて影響を受ける。

(4) すべての算術演算のフラグに影響を与える (AFとCFが意味を持つ)

表3-4の命令は、8086のすべての算術演算のフラグに影響を与える。ただし、AFとCFの値だけが意味を持つ。オーバーフロー、ゼロ、パリティ、サインの値は不明である。

表3-1 フラグ・レジスタに何の影響も与えない命令

CALL	LOOP 命令
CBW	MOV
CWD	MOVS
ESC	NOP
HLT	NOT
IN	OUT
Jump-on-Conditions (条件付分岐)	POP
JCXZ	PUSH
JMP	PUSHF
LAHF	REP
LDS	RET
LEA	SEG
LES	STOS
LOCK	WAIT
LODS	XCHG
	XLAT

表3-2 すべての算術演算フラグに影響する命令

ADC	NEG
ADD	SBB
CMP	SCAS
CMPS	SUB

表3-3 CFを除くすべての算術演算フラグに影響する命令

DEC	INC
-----	-----

表3-4 AFとCFに影響する命令

AAA	AAS
-----	-----

(5) すべての算術演算のフラグに影響を与える（オーバーフローは定義されない）

表3-5の命令は、8086のすべての算術演算のフラグに影響を与える。ただし、オーバーフロー・フラグは意味を持たない。キャリー、補助キャリー、ゼロ、サイン、パリティはすべて意味を持つ。

表3-5 OF を未定義とする命令

DAA	DAS
-----	-----

(6) すべての算術演算のフラグに影響を与える (CFとOFが意味を持つ)

表3-6の命令は、8086のすべての算術演算のフラグに影響を与える。キャリーとオーバーフローのフラグは、通常のように影響を受けない。このフラグの設定についての記述は、その命令を参照のこと。他の算術演算のフラグはすべて不定となる。

表3-6 CF と OF が意味を持ち、すべての算術演算フラグに影響する命令

IMUL	MUL
------	-----

(7) すべての算術演算のフラグに影響を与える (AFは定義されない)

表3-7の命令は、8086のすべての算術演算のフラグに影響を与える。キャリーとオーバーフローは0にクリアされ、AFは不定となる。ゼロ、パリティ、サインは通常どおり設定される。

表3-7 AF を未定義として、すべての算術演算フラグに影響する命令

AND	SHR
OR	TEST
SAR	XOR
SHL	

(8) CFとOFにだけ影響を与える

表3-8の命令は、キャリーとオーバーフローのフラグにだけ影響を与える。補助キャリー、ゼロ、サイン、パリティのフラグは変化しない。

表3-8 CF と OF のみに影響する命令

RCL	ROL
RCR	ROR

(9) 特定のフラグにだけ影響を与える

表3-9の命令は、特定のフラグに作用するために用いられる。たとえば、STIはインタラプト・フラグを1に設定するために用いられる。

表3-9 特定のフラグに影響する命令

CLC (クリア・キャリー)	SAHF (AHを8080フラグへ移動)
CLD (クリア・ディレクション)	STC (セット・キャリー)
CLI (クリア・インタラプト)	STD (セット・ディレクション)
CMC (コンプリメント・キャリー)	STI (セット・インタラプト)

(10) パリティ、サイン、ゼロのフラグに影響を与える

表3-10の命令は、パリティ、サイン、ゼロのフラグに影響を与える。キャリー、オーバーフロー、補助キャリーのフラグは、この命令の実行によって不定となる。

表3-10 PF, SF, ZFに影響する命令

AAD

AAM

(11) すべての算術演算のフラグを不定にする

表3-11の命令は、すべての算術演算のフラグを不定にする。

表3-11 算術演算フラグをすべて未定義にする命令

DIV

IDIV

(12) すべてのフラグをスタックから再現する

表3-12の命令は、スタックからのデータを8086のすべてのフラグにポップする。

表3-12 スタックからすべてのフラグを復元する命令

IRET

POPF

(13) IFとTFにだけ影響を与える

表3-13の命令は、インタラプトとトラップのフラグをクリアする。INTO命令は、オーバーフロー・フラグが1の場合だけ、上記フラグに影響を与える。

表3-13 IFとTFをクリアする命令

INT

INTO

DIVとIDIVの命令は、除算エラーの場合だけIFとTFに影響を与える。

3.4 8086のアドレッシング・モード

8086のアドレッシング・モードに関して、興味深い2つの主要な話題がある。

1. メモリ・アドレスがどのように形成されるか。
2. どのようなアドレッシング・モードが利用可能か。

8086のメモリ・アドレスは、セグメント・レジスタの内容と有効メモリ・アドレスを加えることによって計算される。有効メモリ・アドレスは、他のマイクロプロセッサの場合と同じように、種々のアドレッシング・モードで計算される。選択されたセグメント・レジスタの内容は4ビット左へシフトされ、有効メモリ・アドレスに加えられて、次のように実アドレスの出力を生成する。

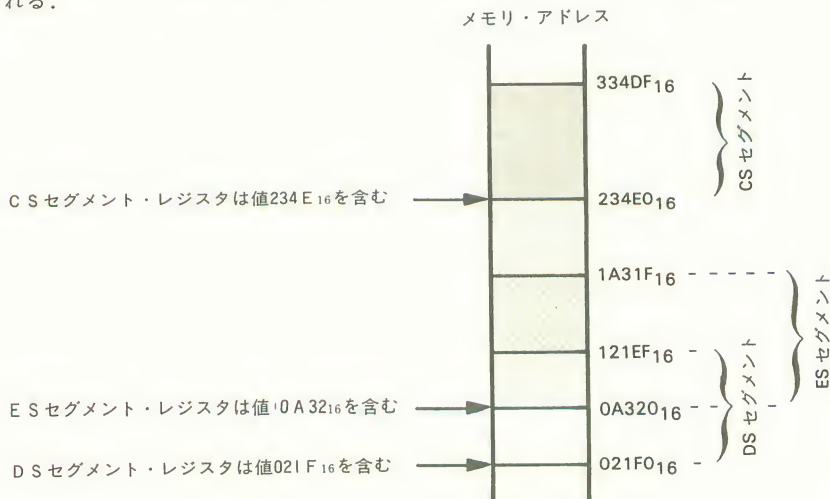
$$\begin{array}{rcl}
 \text{セグメント・レジスタの内容:} & & \text{XXXXXXXXXXXXXXXXXXXX0000} \\
 \text{有効メモリ・アドレス:} & & \text{+0000YYYYYYYYYYYYYYYY} \\
 \hline
 \text{実アドレスの出力:} & & \text{ZZZZZZZZZZZZZZZZZZYYYY}
 \end{array}$$

X, Y, Z は2進数の1桁を表わす。

したがって20ビットのメモリ・アドレスが計算される。これは、1,048,576 バイトの外部メモリを直接にアドレス指定することを可能にする。

8086のアドレスは、したがって、セグメント・アドレスと呼ばれるセグメント・レジスタの内容と、オフセット・アドレスと呼ばれる有効メモリ・アドレスの、2つの異なるアドレスで構成されている。

8086のセグメント・レジスタは、他のマイクロプロセッサのレジスタとは異なっている。これは、丁度16バイトの倍数となるアドレス境界に位置する任意のメモリ位置を示すベース・レジスタとして作用する。適当なメモリ・アドレスを用いると、これは以下のように示される。



上に示したように、各セグメント・レジスタは、65,536バイトのメモリ・セグメントの始まりを識別する。8086は4つのセグメント・レジスタを有するので、常に4つの選ばれた65,536バイトのメモリ・セグメントが存在する。実アドレスの出力は、常にこの4つのうちの1つのセグメント内のメモリ位置を選択する。たとえば、実アドレスの出力がDSセグメント・レジスタと有効メモリ・アドレスの和ならば、実アドレスの出力はDSセグメント内のメモリ位置を選択しなければならない。すなわち、前の例では 021F0₁₆ から 121E F₁₆の範囲のアドレスとなる。同様に、CSセグメント・レジスタと有効メモリ・アドレスの和である実アドレスの出力は、前の例では 234E0₁₆ から 334D F₁₆の範囲のアドレスに位置するCSセグメント内のメモリ位置を選択しなければならない。

セグメント・レジスタの内容についての制限はない。したがって、8086のメモリは、65,536バイトのページに分割されたり、4つのセグメント・レジスタが重なりのないメモリ領域を指定しなければならないことはない。各セグメント・レジスタは、アドレス指定の可能なメモリ内の任意の場所に65,536バイトのメモリ・セグメントの原点を指定し、他のセグメントとの重なりがあってもなくてもよい。

8086のアドレッシング・モードは次の2つの異なるタイプに分けられる。

1. プログラム・メモリ・アドレッシング・モード
2. データ・メモリ・アドレッシング・モード

この2点について論じ、この節の終わりにこれが8086でどのように実行されているかを示す。

3.4.1 プログラム・メモリ・アドレッシング・モード

命令のフェッチが行なわれるときは常に、命令がフェッチされるメモリ位置のアドレスは、プログラム・カウンタ（PCレジスタとも呼ばれる）から得られるオフセットとCSレジスタから得られるセグメントの和として計算される。通常は、命令が実行されるに従って、PCレジスタの内容は増加する。ただし、ジャンプとコールの命令は、以下の3つのうちのいずれかの方法でPCレジスタの内容を変更する。

1. プログラム相対アドレッシング：イミディエイト・データの形で命令によって与えられる8ビットあるいは16ビットのディスプレイメントを、符号付き2進数としてPCレジスタに加算する。これはCSレジスタの内容を変えない。したがって、これをセグメント内操作と呼ぶ。
2. ダイレクト・アドレッシング：イミディエイト・データの形で命令中に存在する2つの新しい16ビットのアドレスを、プログラム・カウンタとCSレジスタにロードする。これはセグメント間操作と呼ばれる。
3. インダイレクト・アドレッシング：データ・メモリ・アドレッシングを選択して（これは次に述べる）、データ・メモリからデータを読み出すのに用いられる。ただし、データ入力、ジャンプあるいはコールの命令によってメモリ・アドレスとして解釈される。2つのインダイレクト・アドレッシングの選択がある。1つの16ビット・データ・ワードが読み出される場合、プログラム・カウンタにロードされて、ジャンプ

あるいはコールは現在のCSセグメント内のメモリ位置を参照する。2つの16ビット・データ・ワードを読み出すこともできる。第1のものはプログラム・カウンタにロードされて、第2のものはCSセグメント・レジスタにロードされる。したがって、インダイレクト・アドレッシングを用いて、アドレス指定可能なメモリ位置にジャンプあるいはコールすることができる。

3.4.2 データ・メモリ・アドレッシング・モード

8086は広範な種類のアドレッシングを持つ。これを次の6つの基本的分類に要約する。

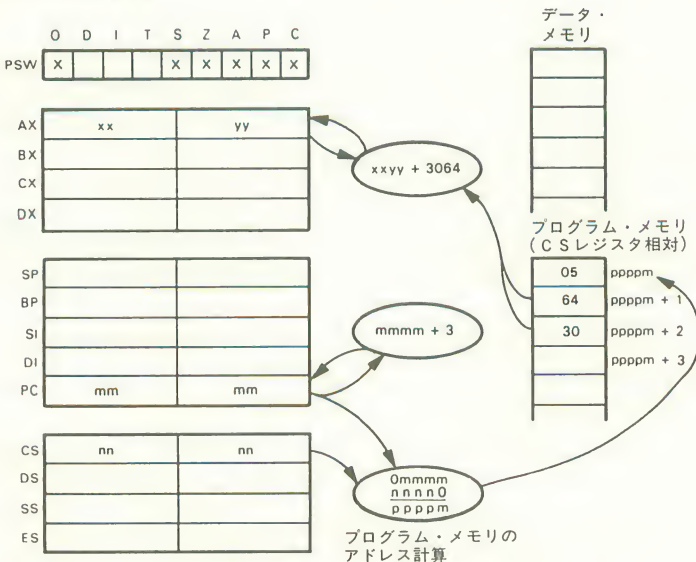
1. イミディエイト
2. ダイレクト
3. ダイレクト・インデックス修飾
4. 暗黙指定
5. ベース相対
6. スタック

(1) イミディエイト・メモリ・アドレッシング

このアドレッシングの形式では、オペランドの1つは命令のオブジェクト・コード（オペコード）直後のバイトに存在する。オペコードの次にアドレッシング・バイトがあれば、イミディエイト・データはそれに続いている。たとえば、

ADD AX, 3064H

はアセンブラに、AXレジスタに3064₁₆を加えるADD命令を生成することを求める。これは次のように示される。



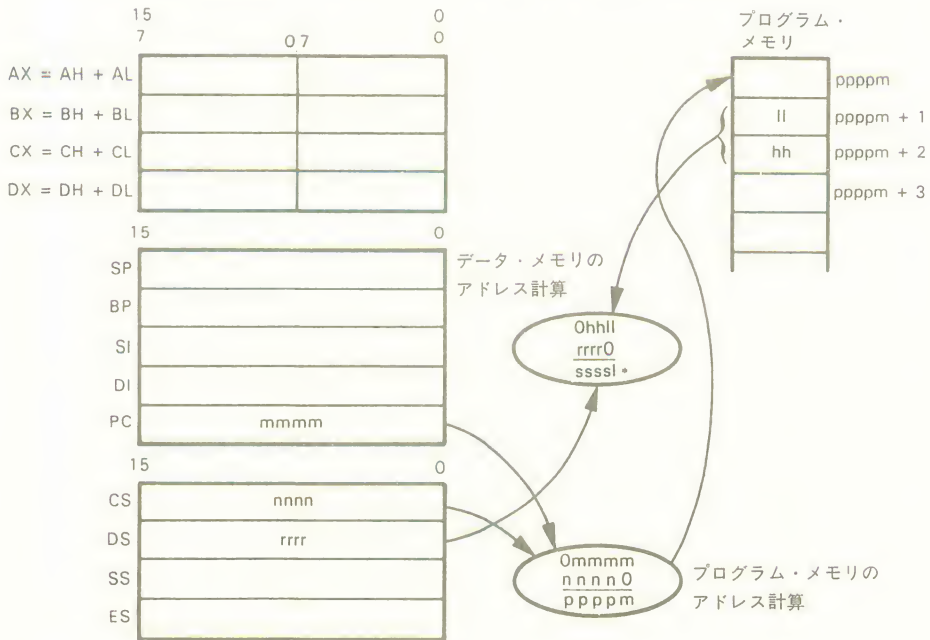
x, y, m, p, n はすべて16進数を表わす

16ビットのイミディエイト・データは、プログラム・メモリにストアされているとき、下位のバイトが上位のバイトに先行していることに注意。これは、8080 Aがイミディエイト・オペランドをプログラム・メモリにストアする方法と一致している。さらに、8086が16ビットのオペランドをデータ・メモリにストアする方法と一致している。16ビットのストアが行なわれるときは、データの下位8ビットが下位のメモリ・バイトにストアされ、データの上位8ビットが続くメモリ・バイトにストアされる。

この例では、AXへのADDの命令のオペコードにすぐ続く2バイトが、AXレジスタに加算される。

(2) ダイレクト・メモリ・アドレッシング

8086は、オブジェクト・コードの2バイトで与えられる16ビットのディスプレイスメントをデータ・セグメント・レジスタに加えて、容易にダイレクト・メモリ・アドレッシングを行なう。上記の和が実メモリ・アドレスとなる。これは次のように示される。



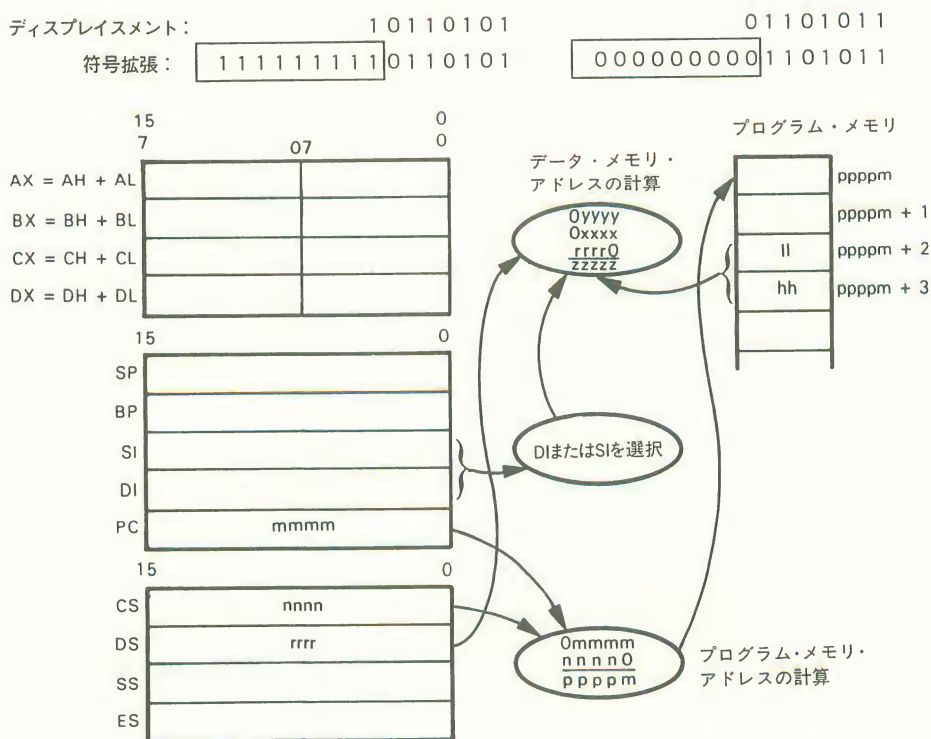
* ダイレクト・メモリ・アドレッシングに対する実際のデータ・メモリ・アドレス

16ビットのアドレス・ディスプレイスメントは、プログラム・メモリに格納されているとき、下位のバイトが上位のバイトに先行していることに注意。これは、8080 Aがアドレスをプログラム・メモリに格納する方法に一致している。

DSレジスタは、上の例で示したように、データ・メモリを直接にアドレス指定するときは、セグメントのベース・アドレスでなければならない。

(3) ダイレクト・インデックス修飾メモリ・アドレッシング

ダイレクト・インデックス修飾アドレッシングは、インデックス・レジスタとしてSIあるいはDIのレジスタを指定することによって行なわれる。有効アドレスを生成するために、指定されたインデックス・レジスタの内容に、8ビットあるいは16ビットのディスプレイメントの加算を指定できる。16ビットのディスプレイメントはオブジェクト・コードの2バイトにストアされ、ダイレクト・メモリ・アドレッシングで示したように、ディスプレイメントの下位バイトが上位バイトに先行している。8ビットのディスプレイメントが指定されたならば、16ビットのディスプレイメントを得るために、下位バイトの最上位ビットが上位バイトに拡張される。これを次に示す。

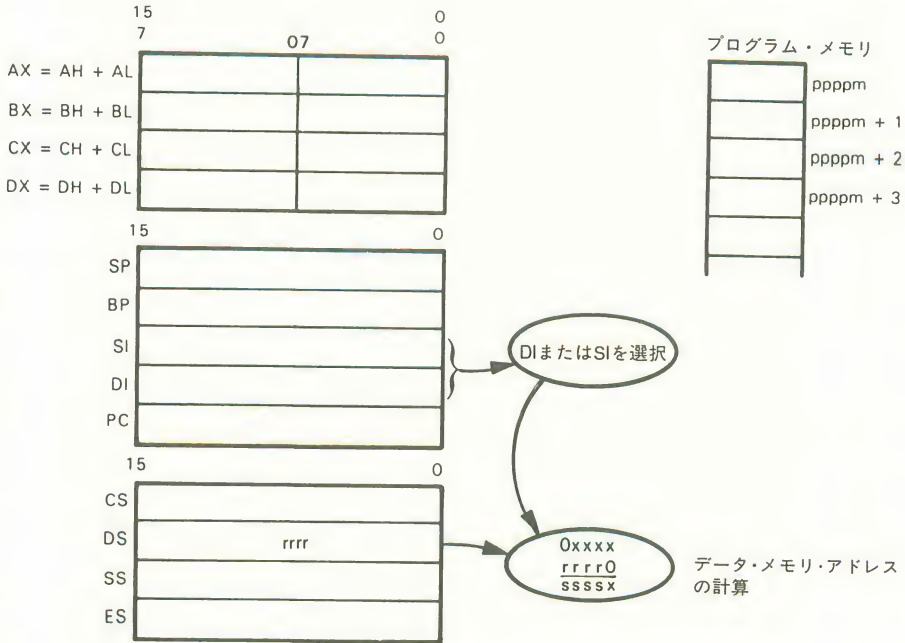


yyyyはプログラム・メモリより得られる16ビットあるいは8ビットのディスプレイメント
xxxxはDIあるいはSIのレジスタから得られるインデックスの値

(4) 暗黙指定メモリ・アドレッシング

8086で暗黙指定メモリ・アドレッシングは、ダイレクト・インデックス修飾メモリ・アドレッシングの縮小されたものとして実行される。ダイレクト・インデックス修飾のアド

レッシング・モードを用いるときにディスプレイメントを指定しなければ、実際は S I あるいは D I のレジスタによるメモリ・アドレッシングの暗黙指定となる。これを次に示す。



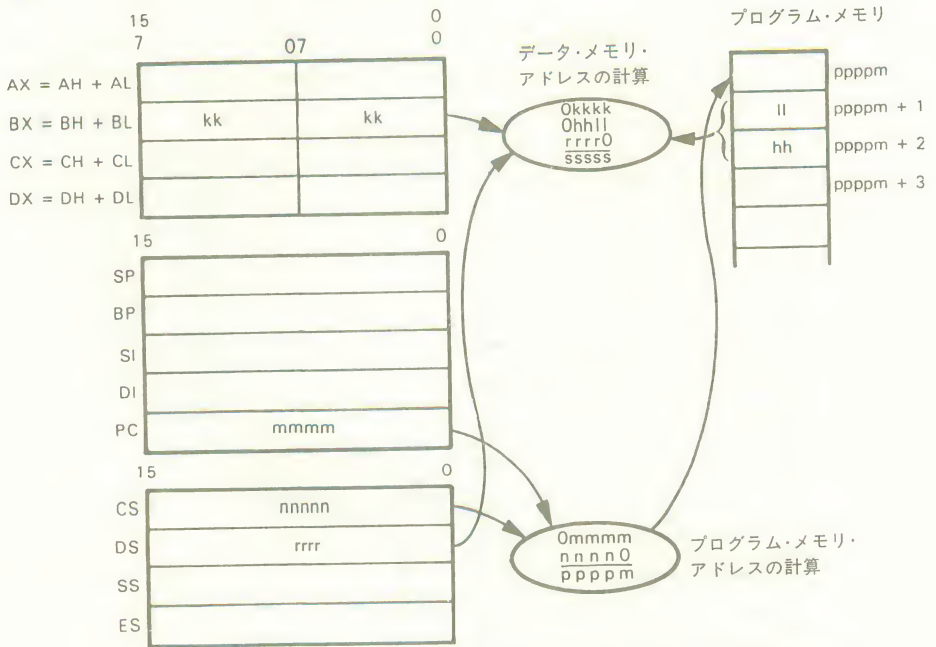
(別に1バイトの命令を実行することによって、DSをCS, SSあるいはESに代えられる。)

(5) ベース相対アドレッシング

8086は、次の2つの方法でベース相対アドレッシングを行なう。

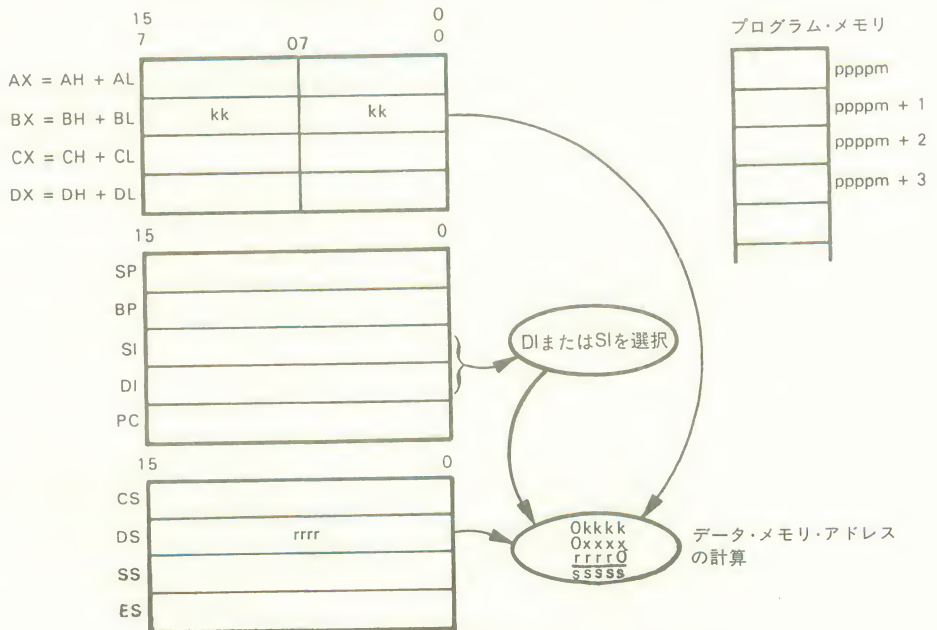
- データ・メモリ・ベース相対アドレッシング。これはDSセグメント（データ・メモリ）内に位置する。
- スタック・ベース相対アドレッシング。これはSSセグメント（スタック・メモリ）内に位置する。

データ・メモリ・ベース相対アドレッシングは、有効アドレスのベースを得るために、BXレジスタの内容を用いる。イミディエイト・アドレッシングを除いて、以上述べたデータ・メモリ・アドレッシング指定のすべては、ベース相対データ・メモリ・アドレッシングと用いることができる。実際、ベース相対データ・メモリ・アドレッシングは、単にBXレジスタの内容を他の方法で得られる有効メモリ・アドレスに加算している。ここで例として、ベース相対ダイレクト・アドレッシングを(A)に示す。



(別に1バイトの命令を実行することによって、DSをCS、ESあるいはSSに代えられる。)

(A) ベース相対ダイレクト・アドレッシング

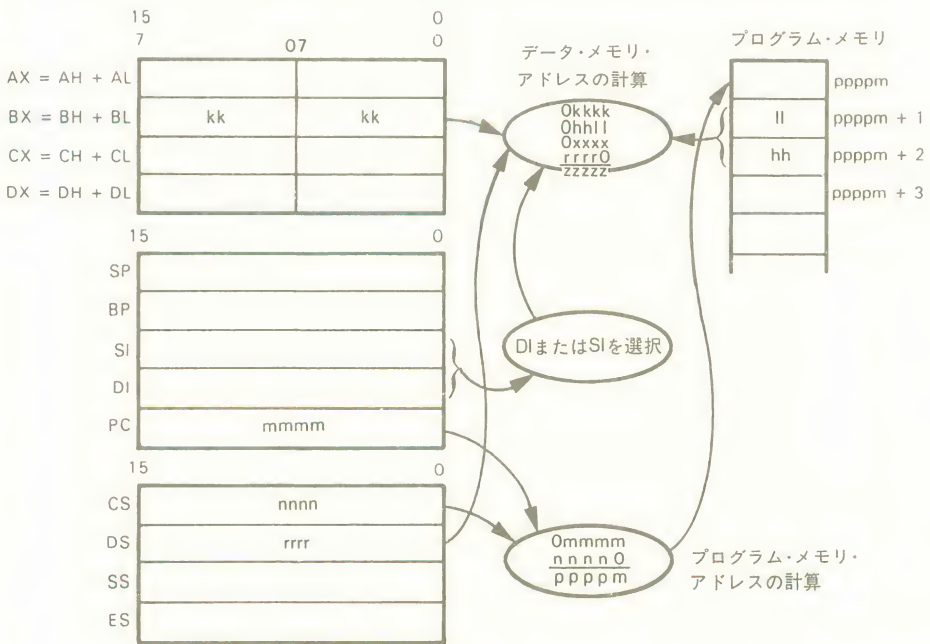


(B) ベース相対暗黙指定アドレッシング

前に述べた単純なダイレクト・アドレッシングは、常に16ビットのディスプレイメントを生成する。ベース相対のダイレクト・アドレッシングでは、前の例で `hhll` と示されている16ビットのディスプレイメントは、符号拡張が行なわれる8ビットのディスプレイメントを持つか、あるいは全くディスプレイメントを持たないことが許されている。

ベース相対の暗黙指定メモリ・アドレッシングは、有効メモリ・アドレスを計算するため、単に `BX` レジスタの内容を選ばれたインデックス・レジスタに加算する。これを(B)に示す。

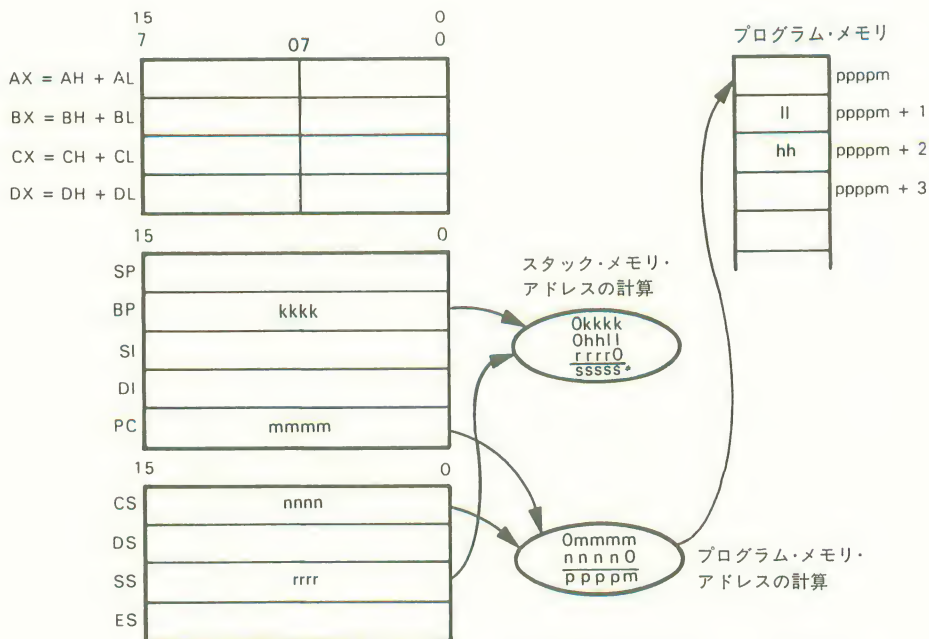
ベース相対のダイレクト・インデックス修飾データ・メモリ・アドレッシングは複雑に見えるが、実はそうではない。`BX` レジスタの内容を、通常のダイレクト・インデックス修飾アドレッシングで計算される有効アドレスに加算する。したがって、ベース相対のダイレクト・インデックス修飾データ・メモリ・アドレッシングは次のように示される。



上の例で、インデックスの値 `xxxx` は選択可能である。ベース相対のダイレクト・メモリ・アドレッシングも利用可能である。この場合は、`SI` と `DI` のレジスタはどちらもアドレス計算に寄与しないので、上の例から `0xxxx` は取り除かれる。

(6) スタック・メモリ・アドレッシング

8086はまた、スタック・メモリ・アドレッシングにも今述べたベース相対のデータ・メモリ・アドレッシングの選択がある。ただしこの場合、ベース・レジスタとして `BP` レジスタが用いられる。ここで例として、ベース相対のダイレクト・スタック・アドレッシングを示す。

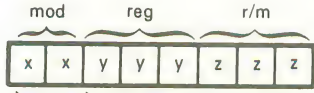


* ベース相対直接メモリ・アドレッシングに対する実際のスタック・メモリ・アドレス出力

上の例では、16ビットのディスプレイメントあるいは符号拡張される8ビットのディスプレイメントとして、ディスプレイメント hhll がある。ベース相対のスタック・メモリ・アドレッシングでは、たとえ 0 であっても、ディスプレイメントを指定する必要がある。

3.4.3 アドレッシング・モード・バイト

8086では、明らかに広範なアドレッシング・モードの中から選択ができる。次に起きる疑問は、このアドレッシング・モードがオブジェクト・コードでどのように実現されるかである。8086は、ほとんどのアドレッシング・モードを、命令のオブジェクト・コードに、アドレッシング・モード・バイトとして知られる1バイトのオブジェクト・コードを用いて指定する。アドレッシング・モード・バイトは、さらに関連のある1あるいは2バイトのディスプレイメントを持てる。最初のオブジェクト・コードの前にプレフィックス命令が含まれていなければ、アドレッシング・モード・バイトは常に命令オブジェクト・コードの2番目のバイトとなる。アドレッシング・モード・バイトは次のように示される。



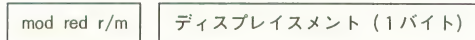
xx は mod フィールドを構成する 2 ビット、mod フィールドはメモリとレジスタのアドレッシングを区別するために用いられる。またメモリ・アドレッシングの場合、アドレッシング・モード・バイトに続くディスプレイスメントのバイト数を指定する。

yyy は reg フィールドを構成する 3 ビット、reg フィールドは操作に用いられるレジスタを定義する。さらに、この 3 ビットは、後で論じられているように、命令を表わすのにも用いられる。

zzz は r/m フィールドを構成する 3 ビット、r/m フィールドは mod フィールドと共にアドレッシングを表わすために用いられる。

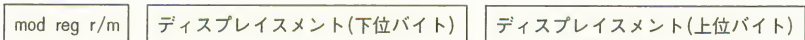
mod =

- 00 メモリ・アドレッシング・モード。r/m は厳密なアドレッシング選択を指定する。ディスプレイスメント・バイトは存在しない。
- 01 メモリ・アドレッシング・モード。r/m は厳密なアドレッシング選択を指定する。1 バイトのディスプレイスメントが存在する。このディスプレイスメント・バイトは、+127 から -128 の範囲の符号付き数値とみなされる。この数値がメモリ・アドレスの計算に用いられるときは、16 ビットに符号拡張される。この場合、アドレッシング・モード・バイトは次のように表わせる。



ここで、mod = 01 でディスプレイスメントは 8 ビットの符号付き数値である。

- 10 メモリ・アドレッシング・モード。r/m はアドレッシング選択を指定する。2 バイトのディスプレイスメントが存在する。最初のバイトはディスプレイスメントの下位 8 ビットで、2 番目のバイトはディスプレイスメントの上位 8 ビットである。この数がメモリ・アドレスの計算に用いられるときは、符号なしの 16 ビット数値として取り扱われる。この場合、アドレッシング・モード・バイトは次のように表わせる。



ここで、mod = 10 である。

- 11 レジスタ・アドレッシング・モード。r/m はレジスタを指定する。w ビットと共に用いられて、8 あるいは 16 ビットのレジスタ選択を決める。

reg 操作に使用されるレジスタの選択に、w ビットと呼ばれるもう 1 つのビットと共に用いられる。命令のオペコードの一部である w ビットは、8 あるいは 16 ビットのどちらが行なわれるかを選択する。

reg	w = 0	w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

1 個のレジスタあるいはメモリのオペランドを必要とする命令 (NOT, NEG など) や、1 個の暗黙指定のオペランドを持つ命令 (MOV イミディエイト, DIV, MUL など) や、目的アドレスの計算にアドレッシング・モードを用いる命令 (JMP, CALL など) は、必要な命令を指定するために、reg フィールドをオペコードのバイトの拡張として用いる。例としては、ADC 命令の項を参照。

r/m 以下に示すように、mod と共にアドレッシング・モードを指定する。

r/m	mod - 00	mod - 01	mod - 10	mod - 11	
				w = 0	w = 1
000	BX + SI	BX + SI + DISP	BX + SI + DISP	AL	AX
001	BX + DI	BX + DI + DISP	BX + DI + DISP	CL	CX
010	BP + SI	BP + SI + DISP	BP + SI + DISP	DL	DX
011	BP + DI	BP + DI + DISP	BP + DI + DISP	BL	BX
100	SI	SI + DISP	SI + DISP	AH	SP
101	DI	DI + DISP	DI + DISP	CH	BP
110	ダイレクト・アドレス	BP + DISP	BP + DISP	DH	SI
111	BX	BX + DISP	BX + DISP	BH	DI

この表は、ダイレクト・アドレスを除いて自明である。mod が 00 で r/m が 110 のとき、オフセット・アドレスは、アドレッシング・モード・バイトに続く 2 バイトから直接に得られる。これは次のように示される。

mod reg r/m	オフセット・アドレス(下位バイト)	オフセット・アドレス(上位バイト)
-------------	-------------------	-------------------

ここで、mod は 00 で r/m は 110 である。

3.4.4 セグメント変更

すべてのアドレッシング・モードは、標準的なデフォルト・セグメント・レジスタを持つ。ほとんどの場合、セグメント変更プレフィックスを用いて、他のセグメント・レジスタを選ぶことができる。プレフィックスを用いるために、デフォルト・セグメント・レジスタの指定を無効にする命令の前に、次のバイトを置く。



rrは命令で用いられるセグメント・レジスタを選択する2ビット

rr=00: ESレジスタ

rr=01: CSレジスタ

rr=10: SSレジスタ

rr=11: DSレジスタ

次の3つの場合、セグメント変更はできない。

1. オフセットの計算にスタック・ポインタ (SPレジスタ) を用いるスタック参照命令 (たとえば, PUSHとCALL) は, 常にセグメント・レジスタとしてSSレジスタを用いる。
2. DIレジスタを用いるストリング命令は, 常にセグメント・レジスタとしてESレジスタを用いる。SIとDIの両方が用いられるストリング操作 (たとえば, MOVSやCMPS) では, セグメント変更プレフィックスが存在するならば, SIのオフセットのセグメント・レジスタを無効にする。
3. セグメント変更プレフィックスは, プログラム・メモリ・アドレッシングと共にには使用できない。すべての命令フェッチは, CSセグメント・レジスタ相対である。

3.4.5 メモリ・アドレッシング・テーブル

メモリ・アドレッシング・モードとメモリ・アドレッシング・バイトの情報を合わせると, 次のようにまとめられる。

r/m =	mod = 00	mode = 01	mod = 10
000	ベース相対インデックス BX + SI	ベース相対ダイレクト・インデックス BX + SI + DISP	ベース相対ダイレクト・インデックス BX + SI + DISP
001	ベース相対インデックス BX + DI	ベース相対ダイレクト・インデックス BX + DI + DISP	ベース相対ダイレクト・インデックス BX + DI + DISP
010	ベース相対インデックス・スタック BP + SI	ベース相対ダイレクト・インデックス・スタック BP + SI + DISP	ベース相対ダイレクト・インデックス・スタック BP + SI + DISP
011	ベース相対インデックス・スタック BP + DI	ベース相対ダイレクト・インデックス・スタック BP + DI + DISP	ベース相対ダイレクト・インデックス・スタック BP + DI + DISP
100	暗黙指定 SI	ダイレクト・インデックス SI + DISP	ダイレクト・インデックス SI + DISP
101	暗黙指定 DI	ダイレクト・インデックス DI + DISP	ダイレクト・インデックス DI + DISP
110	ダイレクト・アドレス	ベース相対ダイレクト・スタック BP + DISP	ベース相対ダイレクト・スタック BP + DISP
111	ベース相対 BX	ベース相対ダイレクト BX + DISP	ベース相対ダイレクト BX + DISP

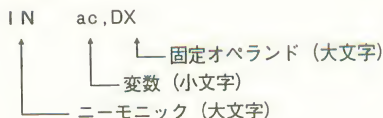
2つのオペランドを持つ命令は、1つのオペランドをメモリから、もう一方のオペランドをCPUのレジスタにしてアクセスすることが非常に多い。また、CPUのレジスタを両方のオペランドとしてアクセスすることも多い。8086は、いくつかの特殊なデータ・ストリング操作命令を除いて、両方のオペランドをメモリとすることは許されていない。次の選択が可能である。

ソース・オペランド	ディスティネーション・オペランド	結果
CPUレジスタ	CPUレジスタ	CPUレジスタ
メモリ位置	CPUレジスタ	CPUレジスタ
CPUレジスタ	メモリ位置	メモリ位置

3.5 命令セットのニーモニック

次の節で、8086アセンブリ言語の各命令について述べる。記述の形式は、次の6つの異なる部分から成る。

1. 命令のニーモニックとそれに関連した種々のオペランド。変化するオペランドは小文字で表わされている。ニーモニック自身と固定しているオペランドは大文字で表わされている。次に例を示す。



2. 命令動作の記述。
3. コード化された命令の機械語。
4. 命令動作の例。非常に簡単な命令では存在しない場合がある。
5. 命令実行のダイアグラム。命令が8086のフラグ、レジスタ、メモリに与える影響を示す。
6. 命令の使用法の簡単な例や、特定の場合にはより有効となる関連のある命令など、類別した情報を含む注釈部分。

3.5.1 略語

ニーモニックと共に、オペランドに用いられる略語を次に示す。

ac	8ビット操作が指定されればALレジスタを、あるいは16ビット操作が指定されればAXレジスタを表わす。これは、8086アセンブリ言語の命令では、ALあるいはAXと表わされる。
addr	16ビットのオフセット・アドレスと16ビットのセグメント・アドレスから成る8086のアドレスである。普通これは、8086アセンブリ言語の命令で、ラベルで表わされる。
count	1あるいはCLレジスタの内容を表わす。これは、8086アセンブリ言語の命

令では、1あるいはCLで示される。

data	8あるいは16ビットのイミディエイト・データを示す。これは、8086アセンブリ言語のステートメントで、自由な数値表現あるいは式として用いられる。
disp	ジャンプと条件付きジャンプで用いられる8ビットの符号付き2進数のディスプレイスメントを示す。常にこれは、8086アセンブリ言語の命令では、ラベルで表わされる。
disp16	コール、ジャンプ、リターンで用いられる16ビット2進数のディスプレイスメントを示す。コールとジャンプの命令で用いられるとき、これはほとんど常に、ラベルで表わされる。リターン命令は普通、これを表わすために数式を用いる。リターン命令との使用は後で示す。
mem	メモリ・オペランドを示す。オペランドを選ぶためのアドレッシング・モードは、アドレッシング・モード・バイトで指定される。これは一般に、アセンブラが適当なアドレッシング・モード・バイトを選ぶラベル、あるいは特定のアドレッシング・モード・バイトの選択が可能な一連のシンボルで表わされる。
mem/reg	メモリあるいはレジスタのオペランドを示す。memとregの記述を参照。
port	I/Oポートを示す。これは、数値表現あるいは式で表わされる。ポート番号は、 00_{16} から FF_{16} の間でなければならない。
reg	8ビット操作が指定されたときは、AH, AL, BH, BL, CH, CL, DH, あるいはDLのレジスタを、16ビット操作が指定されたときは、AX, BX, CX, DX, SP, BP, SI, あるいはDIのレジスタを表わす。
segreg	CS, DS, ES, あるいはSSのレジスタを表わす。
命令のコード化を示す際に用いられる略語を次に示す。	
c	シフトとローテートの命令において、実行されるシフトあるいはローテートの数となる、1あるいはCLレジスタの内容を選択するために用いられる1ビットを示す。 <div style="margin-left: 40px;"> <p>c = 0 1ビットのシフトあるいはローテートを行なう。</p> <p>c = 1 CLレジスタで指定された回数、シフトあるいはローテートを行なう。</p> </div>
d	操作の行なわれる方向を指定するために用いられる1ビットを示す。
disp	ジャンプと条件付きジャンプの命令で、符号付き2進数のディスプレイスメントとして用いられる8ビットを示す。
jj	イミディエイト・データあるいは16ビットのディスプレイスメントの一部を表わすために用いられる2桁の16進数。
kk	イミディエイト・データあるいは16ビットのディスプレイスメントの一部を表わすために用いられる2桁の16進数。

mod reg r/m 前にこの章で述べた 8 ビットのアドレッシング・モード・バイト.

rrr 3 ビットで、8086の汎用レジスタの 1 つを選択する.

8 ビット操作指定

rrr = 000 : A L
 001 : C L
 010 : D L
 011 : B L
 100 : A H
 101 : C H
 110 : D H
 111 : B H

16ビット操作指定

rrr = 000 : A X
 001 : C X
 010 : D X
 011 : B X
 100 : S P
 101 : B P
 110 : S I
 111 : D I

s イミディエイト・データの、符号拡張を行なうかどうかを示す 1 ビット. イミディエイト・データとの 16 ビット操作が指定されているならば、イミディエイト・オペランドをプログラム・メモリ領域の 1 バイトを用いて表わすことができる. s は次のように解釈される.

s = 0 イミディエイト・データに
 2 バイト必要. 符号拡張は
 行なわれない.
 s = 1 1 バイトのイミディエイト・
 データが存在する. 操作に
 必要な 16 ビットのイミディ
 エイト・データを得るため
 に、イミディエイト・デー
 タ・バイトの最上位ビット
 を符号拡張する.

ss 8086のセグメント・レジスタを選択する 2 ビット.

ss = 00 : E S
 01 : C S
 10 : S S
 11 : D S

v ソフトウェア・インタラプトのベクタ位置を示す 1 ビット. v = 0 ならば、インタラプト・サービス・ルーチンはメモリ位置 0000C₁₆ で指定されるアドレスに位置している. それ以外では、アドレスは後続のバイトによって決定される.

w 8 あるいは 16 ビットの操作の、どちらが行なわれるかを示す 1 ビット.

w = 0 8 ビット操作
 w = 1 16 ビット操作

xxx 3 つのドント・ケア* のビットを示す.

yy 命令で用いられる I/O ポート番号を示す 2 桁の 16 進数.

* don't care : 無関係のもの (訳者注).

以下のシンボルは、命令を用いた例で使用される。

H 数字が16進数として処理されることを指定するために、数字の最後に用いられる。

[] カッコの中の式で示されるメモリ位置の内容を表わすために用いられる。BXレジスタが054A₁₆を含むと仮定する。

[BX]

の表現は、カレント・データ・セグメントで054A₁₆のオフセット・アドレスを持つメモリ位置を参照している。

g, h, j, k, m, n, p, q, r, s, t, u, v, w, x, y, z

すべて16進数の1桁を表わすのに用いる。たとえば、

jkk

は16ビットのデータ要素を表わすために用いられ、

ppppm

は20ビットのアドレスを表わすために用いられる。

E A 有効アドレスを示す。これは、個々の命令によって必要な実行サイクル数の計算に現われる。EAは、アドレッシング・モードの実行サイクルを指定し、次のように加算される必要がある。

ダイレクト・アドレッシング	+ 6 サイクル
ダイレクト・インデックス修飾アドレッシング	+ 9 サイクル
暗黙指定アドレッシング	+ 5 サイクル
ベース相対アドレッシング	+ 5 サイクル
ベース相対ダイレクト・アドレッシング	+ 9 サイクル
ベース相対インデックス修飾アドレッシング	+ 7 あるいは + 8 サイクル*
ベース相対ダイレクト・インデックス修飾アドレッシング	+ 11 あるいは + 12 サイクル*

次の場合さらにアドレッシング・モード・サイクルの加算が必要となる。

セグメント変更プレフィックスがあるとき	+ 2 サイクル
16ビットのワードが指定されて、そのワードが奇数のメモリ・アドレスに位置するとき	+ 4 サイクル

* BP+SI と BX+DI のモードは、BP+DI と BX+SI のモードよりも1つ多くのクロックが必要。

3.6 8086アセンブリ言語の命令 (アルファベット順)

AAA (ASCII Adjust for Addition)

ASCII 加算結果のアジャスト*を行なう。

この命令は、オペランドとして2つのASCIIキャラクタの加算によって結果が得られたものとして、ALレジスタの結果を補正するために用いられる。この補正（アジャストメント）は次のようにして行なわれる。

1. ALレジスタの下位4ビットが0と9の間で、しかもAFフラグが0ならば、ステップ3へ。
2. ALレジスタの下位4ビットがAとFの間、あるいはAFフラグが1ならば、ALレジスタに6を加え、AHレジスタに1を加えて、AFフラグを1にセットする。
3. ALレジスタの上位4ビットをクリアする。
4. CFフラグにAFフラグの値を設定する。

命令コードを次に示す。

$$\frac{\text{AAA}}{37}$$

たとえば、AXレジスタの内容を0535₁₆、BLレジスタの内容を39₁₆とすると、

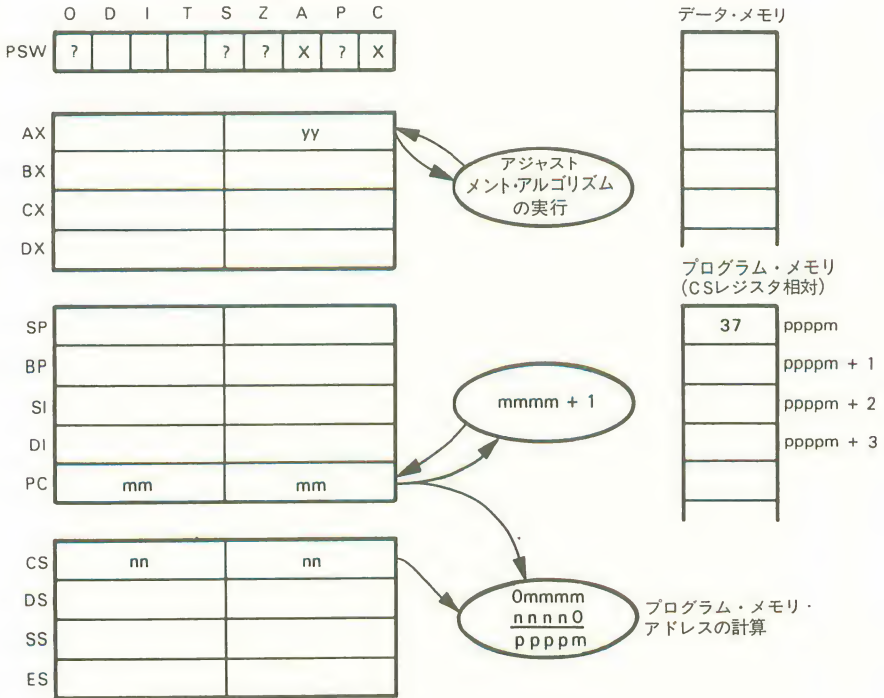
```
ADD  AL,BL
AAA
```

を実行することによって、AXの内容は0604₁₆となる。ADD命令の結果、

$$\begin{array}{r} 35_{16} = 0011 \ 0101 \\ 39_{16} = 0011 \ 1001 \\ \hline 0110 \ 1110 \end{array}$$

となり、6E₁₆がALにストアされる。AAA命令によるアジャストメント・アルゴリズムの結果、AFとCFのフラグは1にセットされ、04₁₆がALレジスタにストアされ、そしてAHレジスタは増加して06₁₆となる。

* 補正, 修正, 調整 (訳者注)。



AAA
サイクル数: 4

注)

1. 2つのオペランドが1桁のBCDである場合にも、この命令が使えることに注意。この種の操作がなぜ必要とされるかについては読者に任せる。
2. 2つのパック形式BCD加算についての補正は、DAA命令を参照。
3. この命令の結果、OF, PF, SF, ZFの各フラグは不定となる。

AAD (ASCII Adjust for Division)

除算のためにAXレジスタのアジャストを行なう。

この命令では、AHとALのレジスタがアンパック形式BCDオペランドを含んでいることを仮定している。この命令は、前記情報をALレジスタの2進数オペランドに変換する。変換のアルゴリズムでは、10の位はAHレジスタに、1の位はALレジスタにあることを前提としている。次にAADのアルゴリズムを示す。

1. AHレジスタの内容に $0A_{16}$ を乗じる。

2. AHをALに加算する.
3. AHレジスタに 00_{16} をストアする.
4. 次のようにフラグを設定する.

CF, OF, AF: 不定

PF, ZF, SF: ALレジスタに基づいて決定

命令コードを次に示す.

AAD
D50A

AXレジスタが 0604_{16} を含むとすると,

AAD

の実行の結果, AXレジスタは 0040_{16} となる. フラグは次のように設定される.

CF, OF, AF: 不定

SF: ALレジスタの最上位ビットが0なので, $SF = 0$

ZF: ALレジスタは0でないので, $ZF = 0$

PF: ALレジスタの1のビットは1個なので, $PF = 0$

	O	D	I	T	S	Z	A	P	C
PSW	?				X	X	?	X	?

AX	xx	yy
BX		
CX		
DX		

xyxyにアジャ
ストメントを実行

SP		
BP		
SI		
DI		
PC	mm	mm

mmmm + 2

CS	nn	nn
DS		
SS		
ES		

0mmmm
nnnn0
ppppm

プログラム・メモリ・
アドレスの計算

データ・メモリ

プログラム・メモリ
(CSレジスタ相対)

D5	ppppm
0A	ppppm + 1
	ppppm + 2
	ppppm + 3

AAD
サイクル数: 60

注)

1. この命令は、除算の ASCII オペランドをアジャストするためにも使用可能。たとえば、AX レジスタが 3537_{16} を含む場合を考えると、

```
AND    AX,0F0FH
AAD
```

の実行後、AX レジスタは 0039_{16} となる。

AAM (ASCII Adjust for Multiplication)

BCD 乗算結果のアジャストを行なう。

この命令は、オペランドとして2つのアンパック形式BCDによる乗算が実行されたことを前提に、AL レジスタの結果を補正する。アジャストメントは次のように行なわれる。

1. AL レジスタを $0A_{16}$ で割り、商をAH レジスタに、余りをAL レジスタにストアする。
2. 以下のようにフラグを設定する。

CF, OF, AF: 不定

PF, SF, ZF: AL レジスタに基づいて決定

命令コードを次に示す。

```
AAM
D40A
```

AL レジスタが 07_{16} を含み、BL レジスタが 09_{16} を含むと仮定すると、

```
MUL    AL, BL
AAM
```

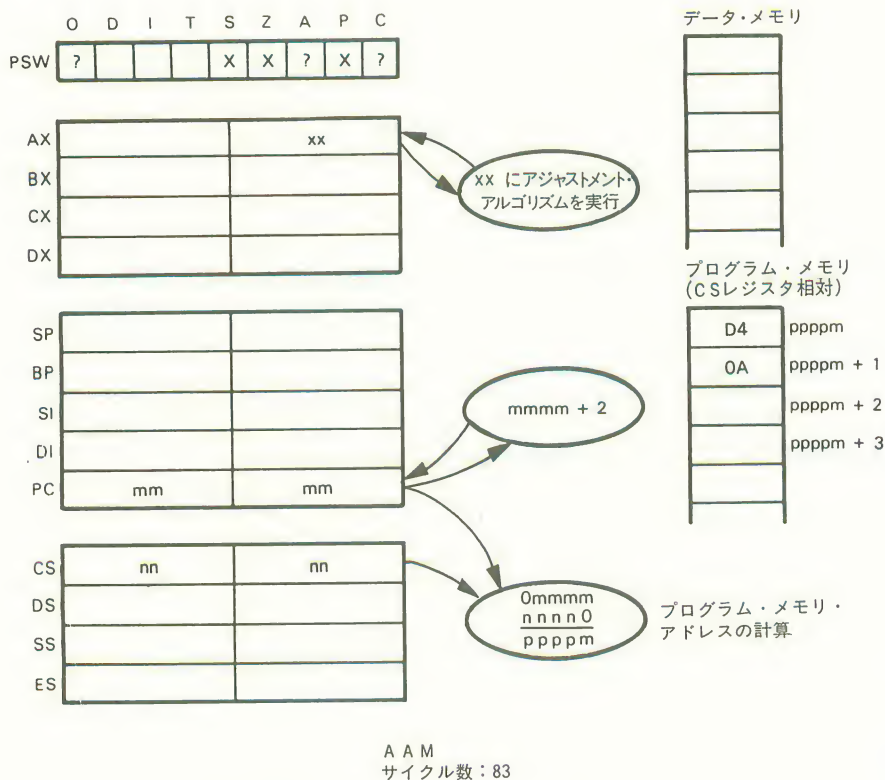
の実行後、AX レジスタは 0603_{16} となる。MUL 命令の結果、 $3F_{16}$ がAL レジスタにストアされる。アジャストメント・アルゴリズムを行なうことによってAX レジスタは 0603_{16} となり、フラグは次のように設定される。

CF, OF, AF: 不定。

SF: AL レジスタの最上位ビットが0なので、SF = 0。

ZF: AL レジスタが0でないので、ZF = 0。

PF: AL レジスタに1のビットが2個あるので、PF = 1。



AAS (ASCII Adjust for Subtraction)

ASCII 減算結果のアジャストを行なう。

この命令は、オペランドとして2つのASCIIキャラクタによる減算が実行されたことを前提に、ALレジスタの結果を補正する。アジャストメントは次のように行なわれる。

1. ALレジスタの下位4ビットが0と9の間で、しかもAFフラグが0ならば、ステップ3へ。
2. ALレジスタの下位4ビットがAとFの間、あるいはAFフラグが1ならば、ALレジスタから6を引き、AHレジスタから1を減じて、AFフラグを1にセットする。
3. ALレジスタの上位4ビットをクリアする。
4. CFフラグにAFフラグの値を設定する。

命令コードを次に示す。

AAS
3F

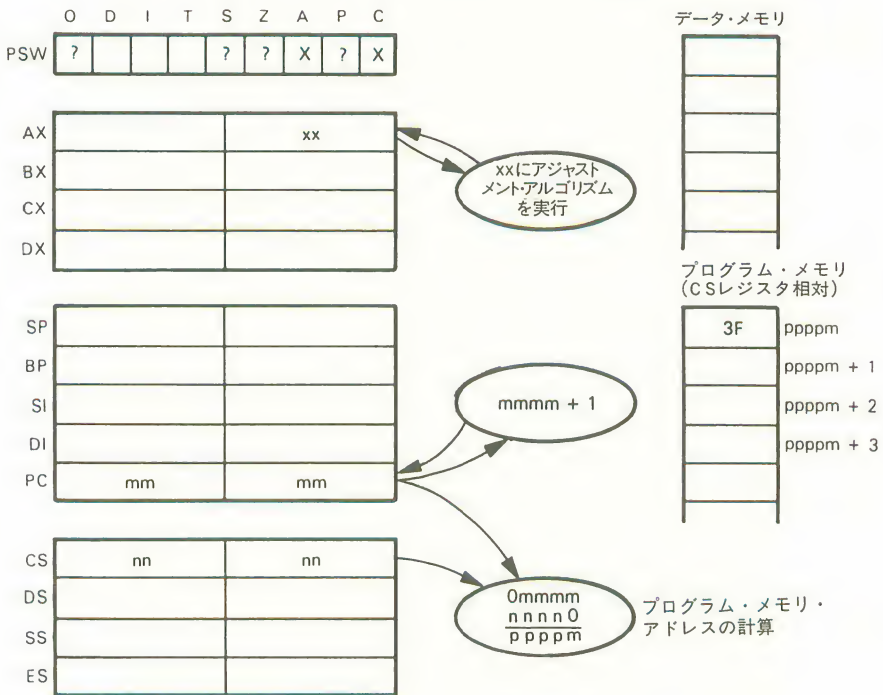
たとえば、AXレジスタが0438₁₆を含むと仮定する。

SUB AL, 35H
AAS

の実行後、AXレジスタは0403₁₆となる。SUB命令の結果、

38₁₆ = 0011 1000
35₁₆の2の補数 = 1100 1011
0000 0011

03₁₆がALにストアされる。AAS命令のアジャストメント・アルゴリズムの実行では、この場合AXレジスタは変更されない。AFとCFのフラグは0に設定される。



AAS
サイクル数: 4

注)

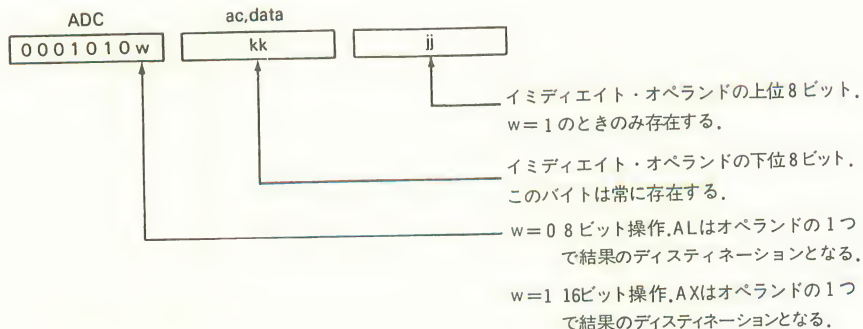
1. ASCII 加算結果のアジャストについては、AAA命令参照。パック形式BCDの加算と減算の結果のアジャストについては、DAAとDASの命令を参照。
2. この命令の実行により、PF、ZF、SF、OFの各フラグの値は不定となる。

ADC ac,data (Add with Carry)

A XあるいはA Lのレジスタに、キャリーと共にイミディエイト・データを加算する。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データとキャリーを、A L（8ビット操作）あるいはA X（16ビット操作）のレジスタに加算するために用いられる。

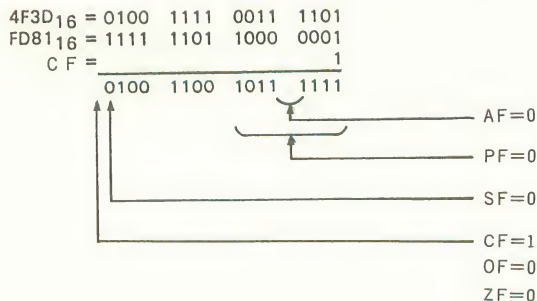
命令コードを次に示す。

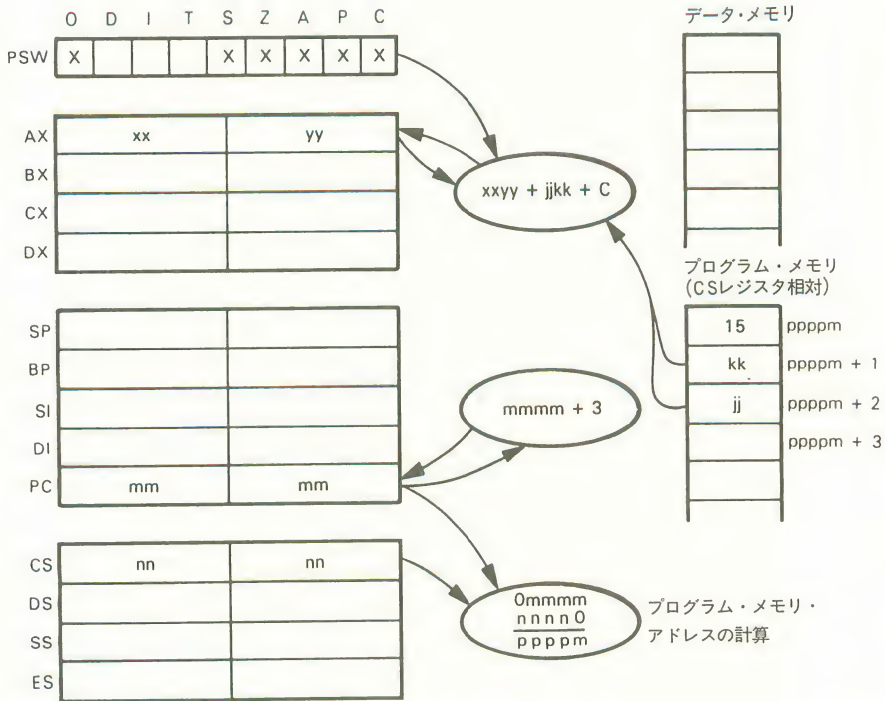


たとえば、A Xレジスタが4F3D₁₆を含み、キャリーが1の場合を考える。

ADC AX,0FD81H

の実行後、A Xレジスタは4CBF₁₆を含み、キャリーは1となる。





ADC AX, jkk
サイクル数: 4

注)

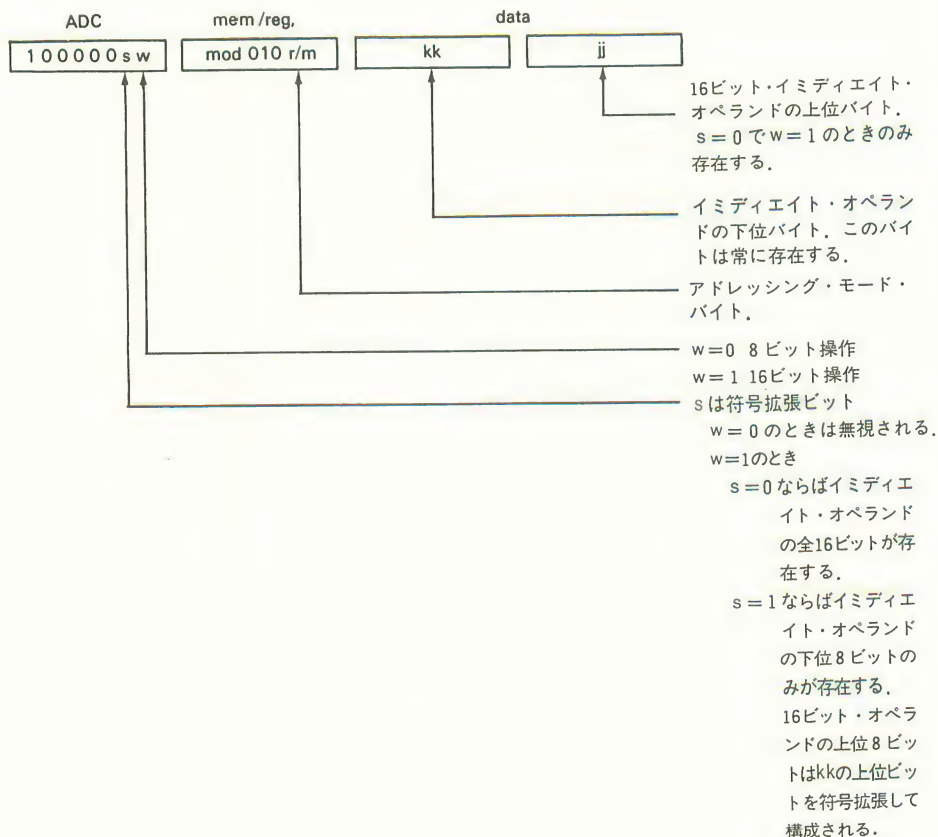
1. この命令は、8080の命令 `ACI data` と同じ機能を果たす。さらに、この命令はキャリーと共にイミディエイトの16ビットの加算を行なえる。

ADC mem/reg, data (Add with Carry)

レジスタあるいはメモリに対して、キャリーと共にイミディエイトを加算する。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データとキャリーを、指定されたレジスタあるいはメモリに加算するために用いられる。8ビットあるいは16ビットの操作が指定できる。

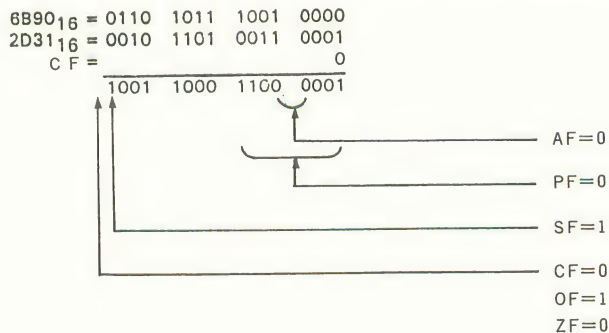
命令コードを次に示す。



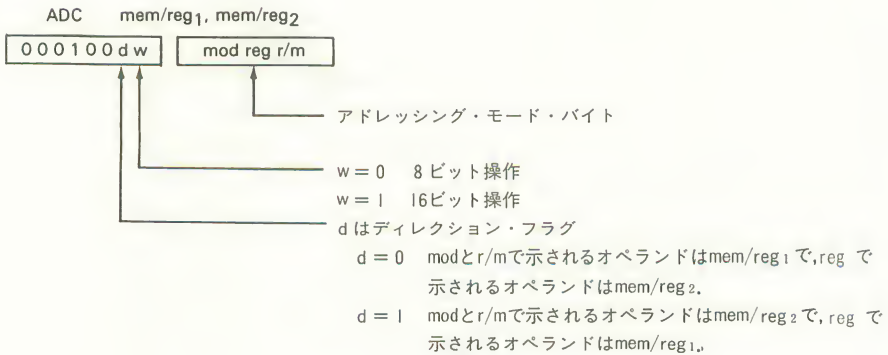
Dレジスタが $E400_{16}$ を含み、S Iレジスタが 0040_{16} を含み、メモリ位置 $E4040_{16}$ のワードが $6B90_{16}$ で、キャリイは0であると仮定する。

ADC [SI], 2D31H

の実行後、メモリ位置 $E4040_{16}$ のワードは $98C1_{16}$ を含み、キャリイは0となる。



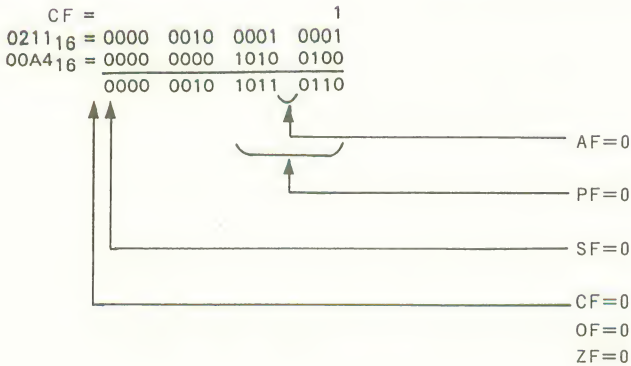
命令コードを次に示す.

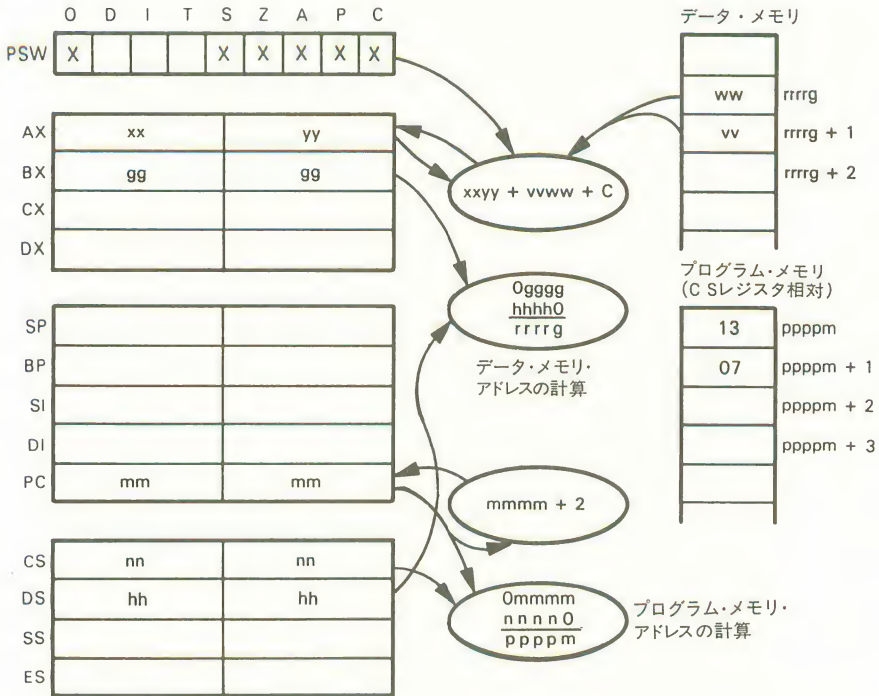


A Xレジスタが 0211_{16} を含み, B Xレジスタが 0084_{16} を含み, D Sレジスタが $1C00_{16}$ を含み, キャリーが1で, $1C084_{16}$ のメモリ・ワードの内容が $00A4_{16}$ であると仮定する.

ADC AX, [BX]

の実行後, A Xレジスタは $02B6_{16}$ を含み, フラグは次のようになる.





ADC AX, [BX]

サイクル数: メモリからレジスタに対して: $9 + EA$ レジスタからメモリに対して: $16 + EA$

レジスタからレジスタに対して: 3

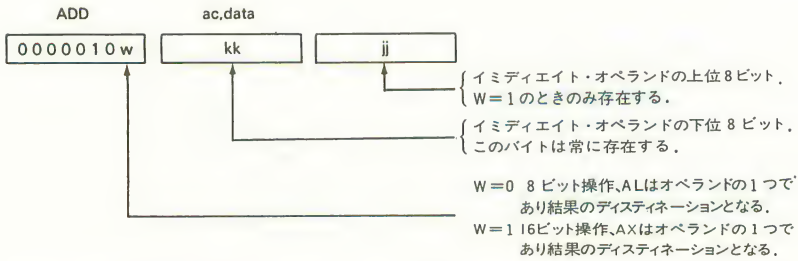
注)

1. この命令は通常 AX あるいは AL のレジスタに対しては用いられない。ADC ac, data 命令はこの機能をより少ないバイト数で行なう。

ADD ac, data (Add)

AX あるいは AL のレジスタにイミディエイト・データを加算する。

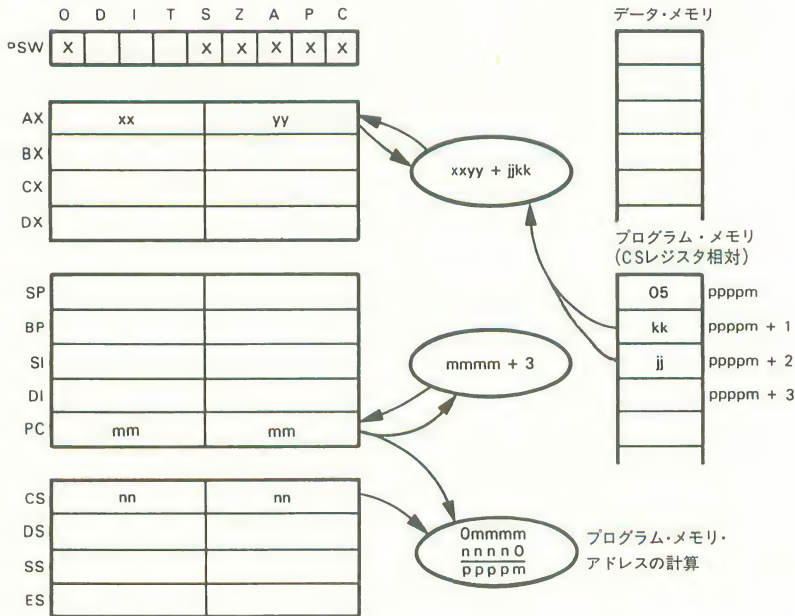
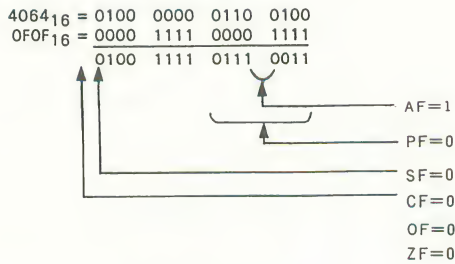
この命令は、後続のプログラム・メモリに存在するイミディエイト・データを、AL (8ビット操作) あるいは AX (16ビット操作) のレジスタに加算するために用いられる。命令コードを次に示す。



A Xレジスタが 4064_{16} を含み、キャリーが 1 であると仮定する。

ADD AX,0F0FH

の実行の結果、A Xレジスタは $4F73_{16}$ を含む。



ADD AX, jkk
 サイクル数: 4

注)

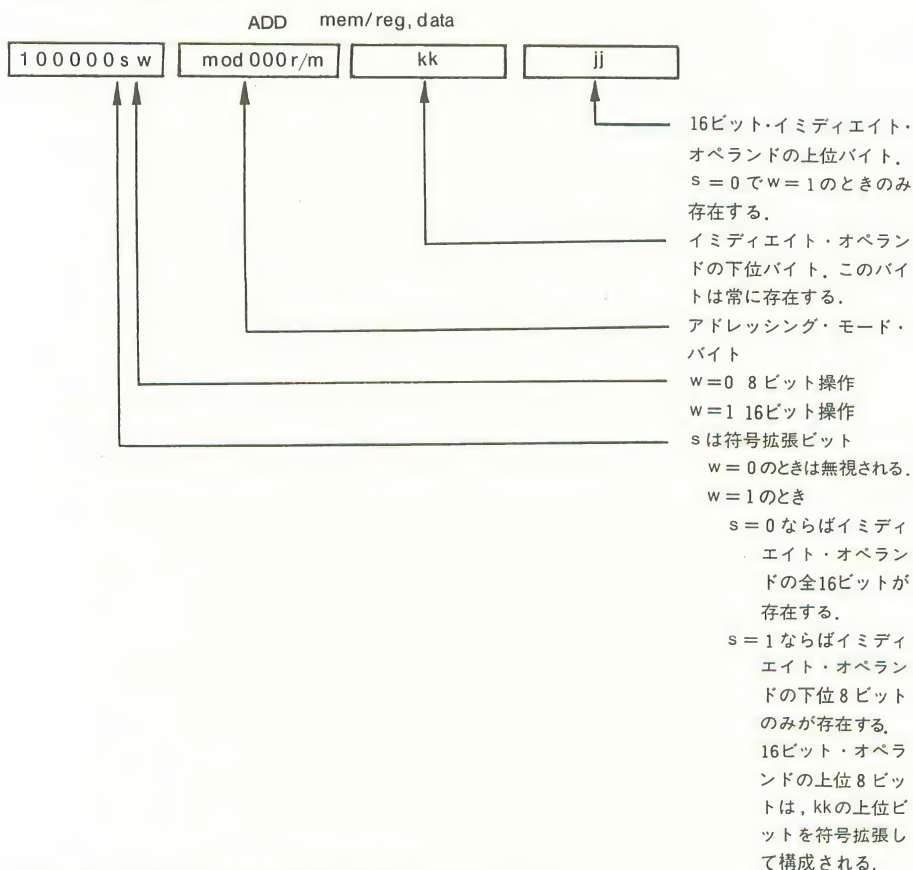
1. この命令は、8080の命令 `ADI data` と同じ機能を果たす。この命令はさらに16ビット・イミディエイト・データの要素を加算することができる。

ADD mem/reg, data (Add)

レジスタあるいはメモリにイミディエイト・データを加算する。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データを、指定されたレジスタあるいはメモリ位置に加算するために用いられる。8あるいは16ビットの操作が指定できる。

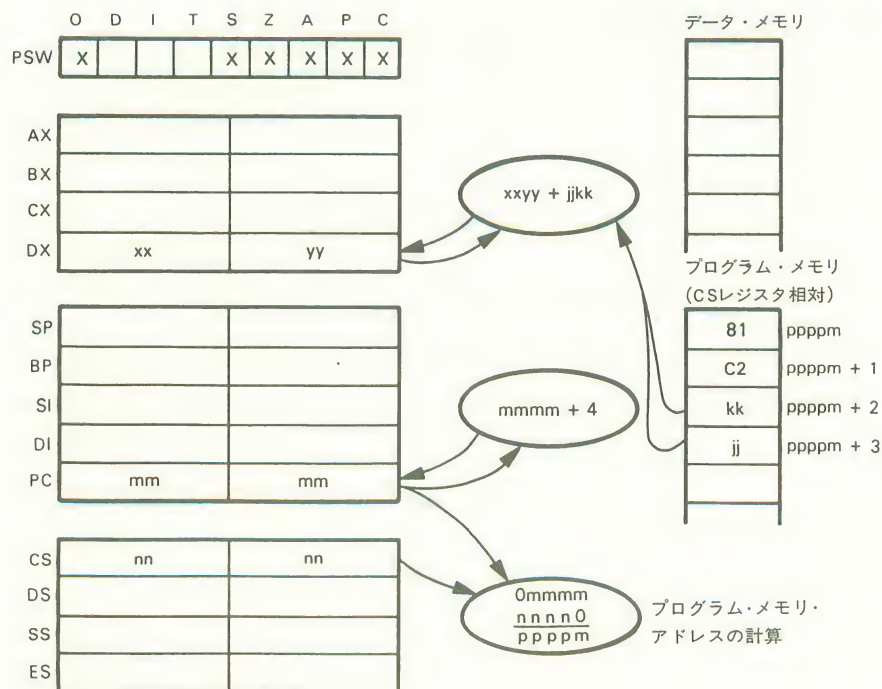
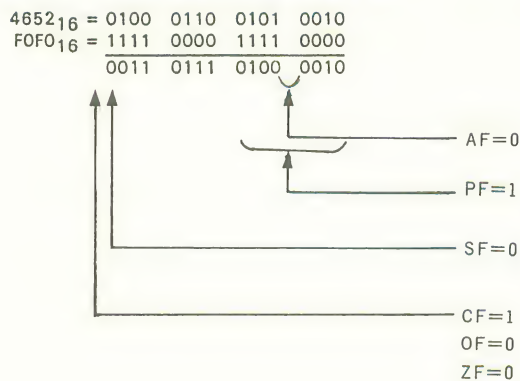
命令コードを次に示す。



たとえば、`DX`レジスタが 4652_{16} を含んでいるとして、

`ADD DX, 0F0F0H`

を実行すると、D Xレジスタの内容は 3742_{16} になる。



注)

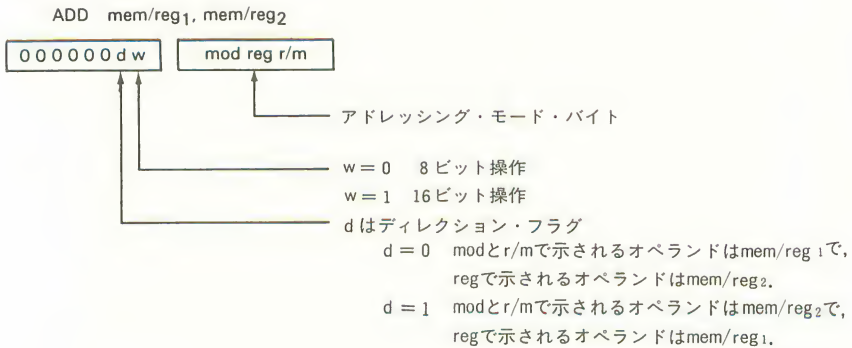
- この命令は通常A XあるいはA Lのレジスタに対しては用いられない。ADD ac, data 命令はこの機能をより少ないバイト数で行なう。

ADD mem/reg₁, mem/reg₂ (Add)

$\left\{ \begin{array}{l} \text{レジスタをレジスタに} \\ \text{レジスタをメモリに} \\ \text{メモリをレジスタに} \end{array} \right\}$ 加算する。

mem/reg₂で指定されるレジスタあるいはメモリ位置の内容を, mem/reg₁で指定されるレジスタあるいはメモリ位置の内容に加える。8あるいは16ビットの操作が指定できる。mem/reg₁あるいはmem/reg₂はメモリ・オペランドとなるが、オペランドの一方はレジスタ・オペランドでなければならない。

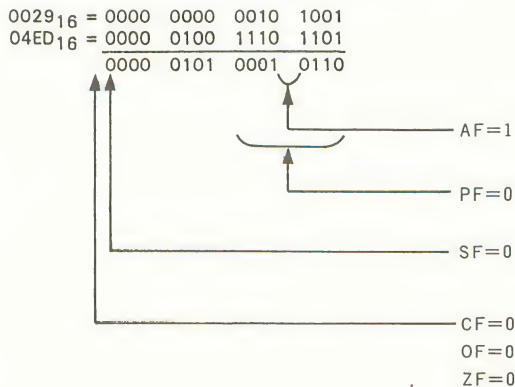
命令コードを次に示す。

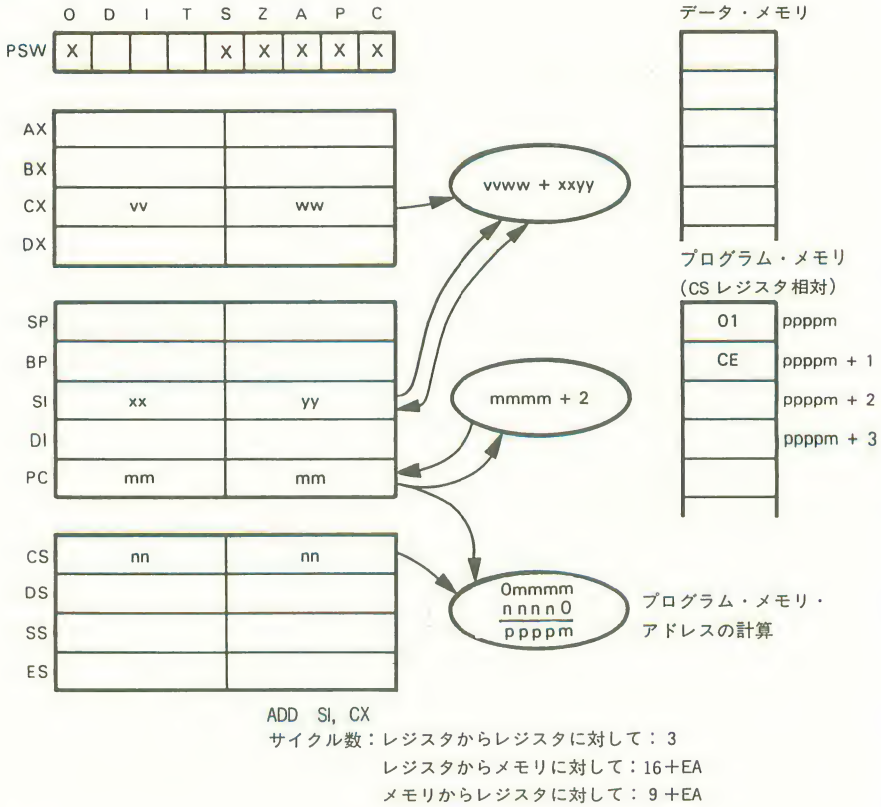


CXレジスタが0029₁₆を含み、SIレジスタの内容が04ED₁₆であると仮定する。

ADD SI, CX

の実行後、SIレジスタの内容とフラグは次のように変化する。



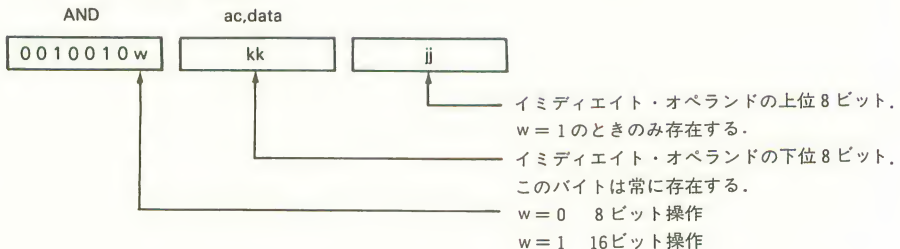


AND ac, data (AND)

ALあるいはAXレジスタとイミディエイト・データのANDをとる。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データと、AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容のANDを行なうために用いられる。

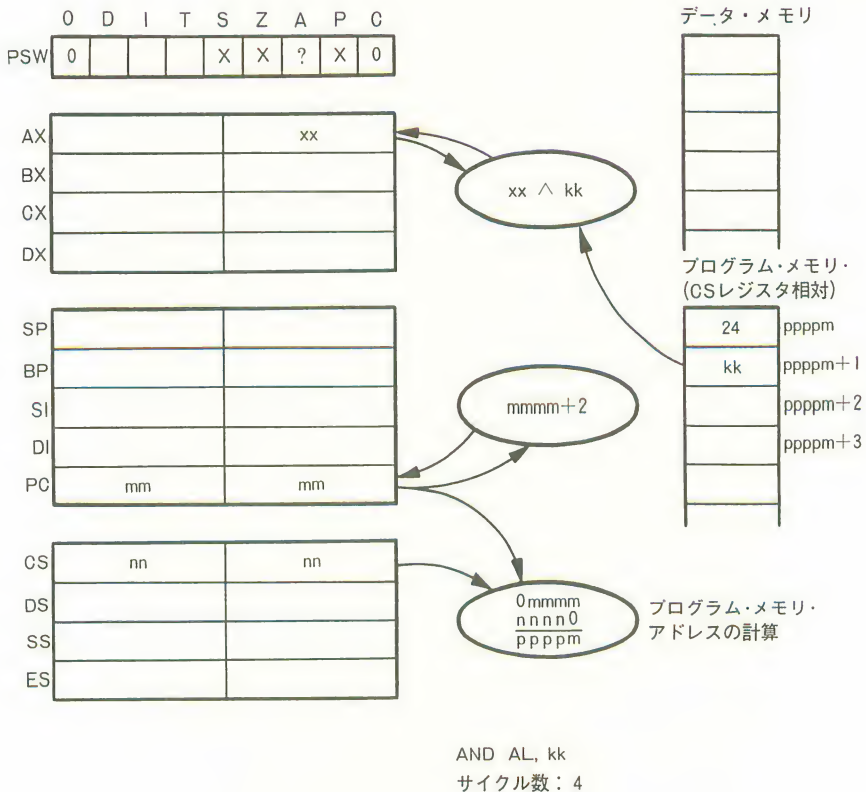
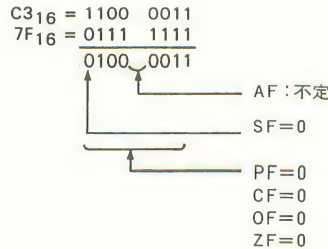
命令コードを次に示す。



例として、ALレジスタがC3₁₆を含む場合を考える。

AND AL,7FH

の実行後、ALレジスタは43₁₆を含む。



注)

1. この命令は、8080の命令 ANI data と同じ機能を果たす。しかし、16ビット操作も

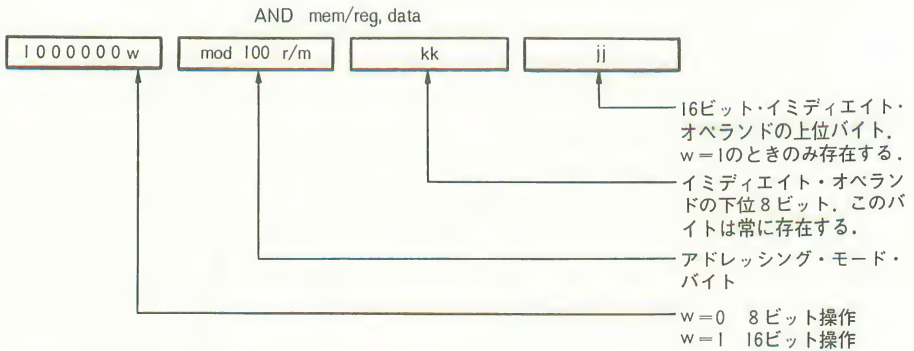
可能である。

2. 他の汎用レジスタあるいはメモリとイミディエイトのANDが必要ならば、AND mem/reg,data 命令を参照。

AND mem/reg,data (AND)

レジスタあるいはメモリとイミディエイト・データのANDをとる。

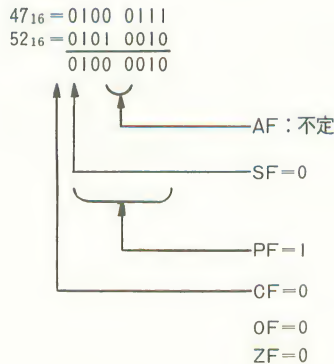
後続のプログラム・メモリ・バイトに存在するイミディエイト・データと、指定されたレジスタあるいはメモリ位置のANDをとる。8あるいは16ビットの操作が指定できる。命令コードを次に示す。

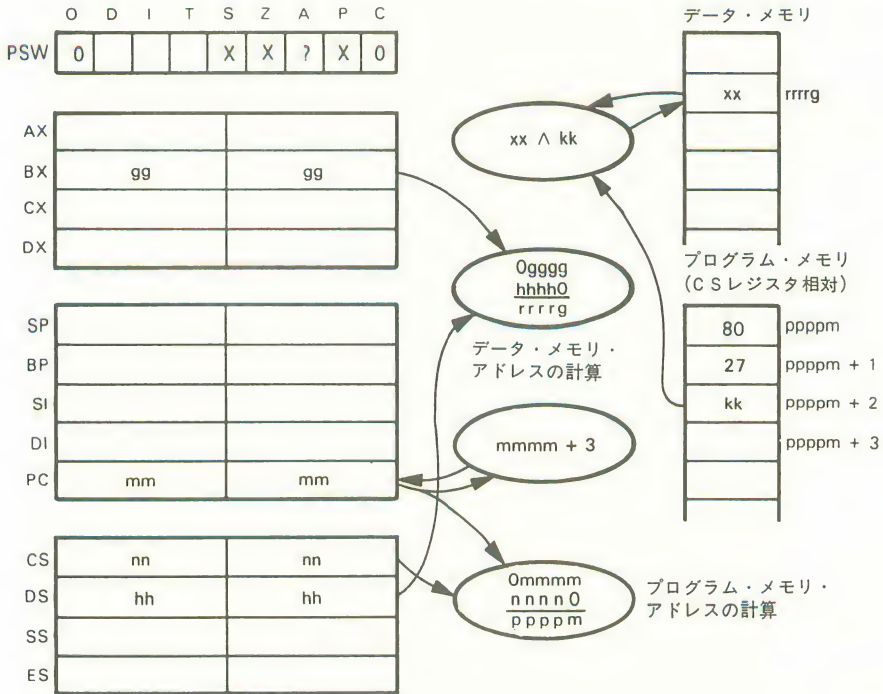


BXレジスタが 0104_{16} を含み、DSレジスタが 0000_{16} を含み、メモリ位置 00104_{16} のバイトが 47_{16} の場合を考える。

AND [BX],52H

の実行後、メモリ位置 00104_{16} は 42_{16} を含む。





AND [BX], kk

サイクル数: メモリに対して: 17+EA

レジスタに対して: 4

注)

1. この命令は通常AXあるいはALのレジスタに対しては用いられない。この機能に対しては、命令 AND ac,data が用いられる。

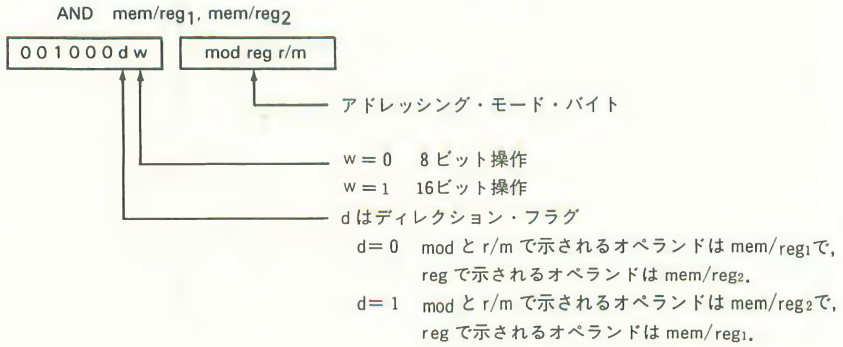
AND mem/reg₁, mem/reg₂ (AND)

レジスタとレジスタ
レジスタとメモリ
メモリとレジスタ

でANDをとる。

mem/reg₂ で指定されるレジスタあるいはメモリ位置の内容と、mem/reg₁ で指定されるレジスタあるいはメモリ位置の内容のANDをとる。8あるいは16ビットの操作が指定できる。mem/reg₁ あるいはmem/reg₂ はメモリ・オペランドとなるが、オペランドの一方はレジスタ・オペランドでなければならない。

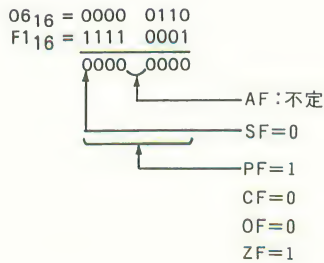
命令コードを次に示す.

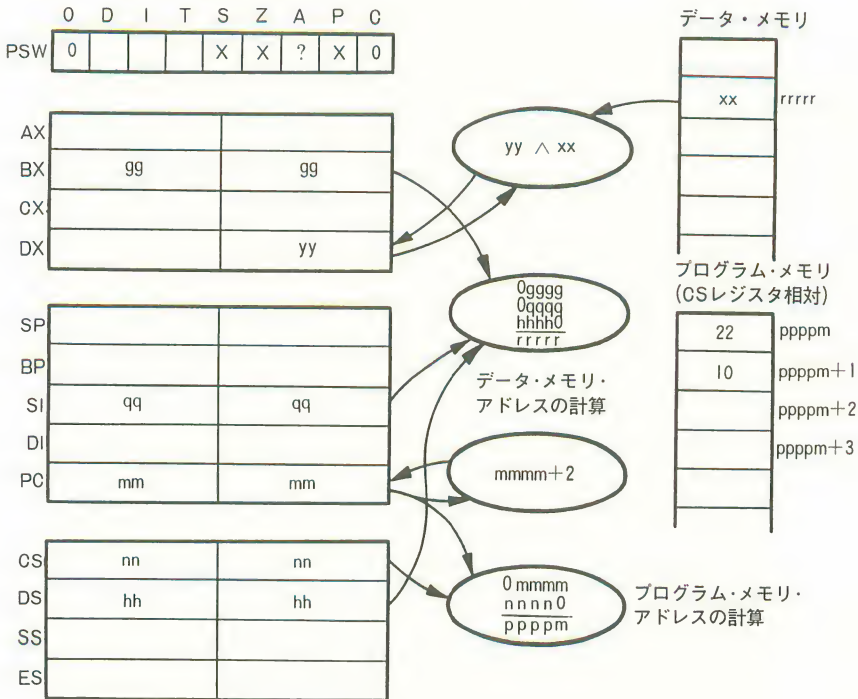


例として, DLレジスタが06₁₆を含み, DSレジスタがB000₁₆を含み, BXレジスタが0010₁₆を含み, SIレジスタが0006₁₆を含み, メモリ位置B0016₁₆のバイトがF1₁₆を含む場合を考える.

AND DL, [BX+SI]

の実行後, DLレジスタは00₁₆を含み, フラグは次のようになる.





AND DL, [BX+SI]

サイクル数: メモリからレジスタに対して: $9 + EA$

レジスタからメモリに対して: $16 + EA$

レジスタからレジスタに対して: 3

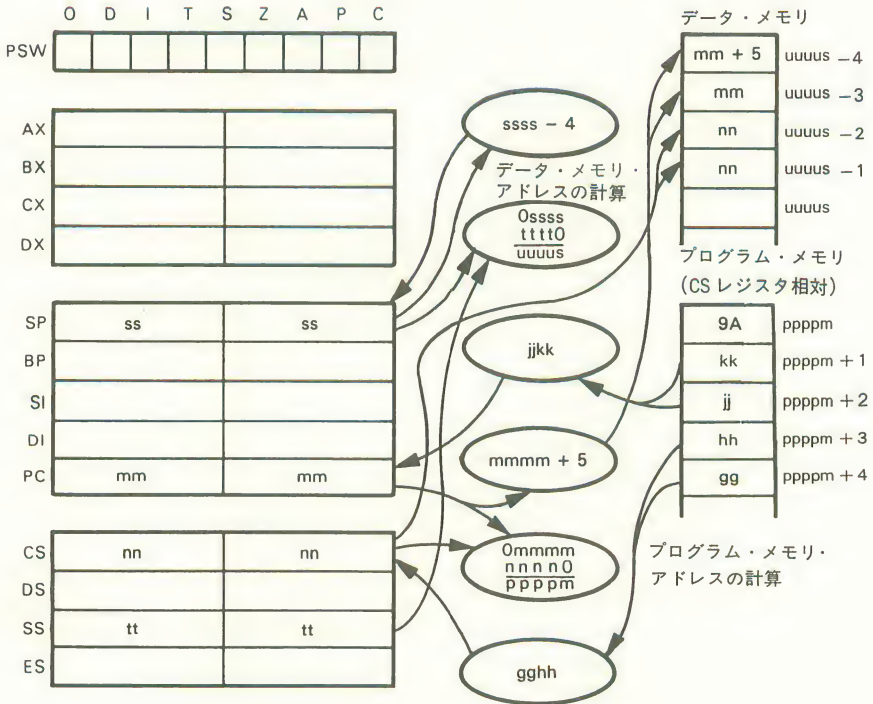
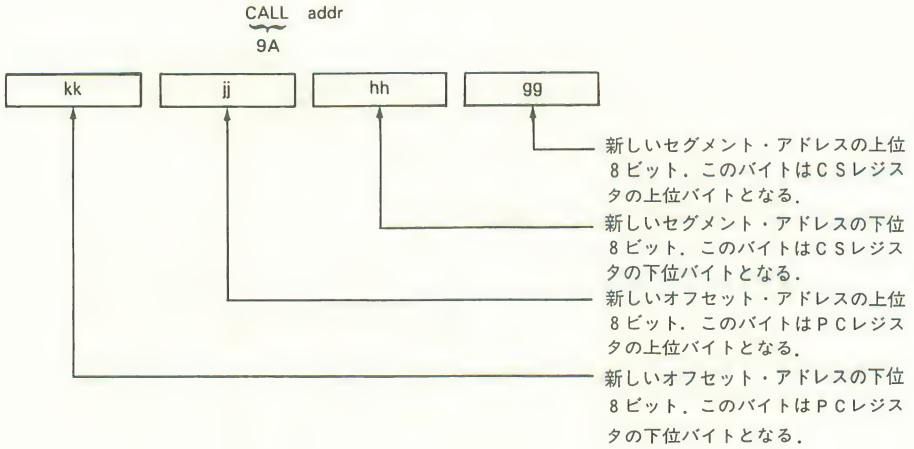
CALL addr (CALL)

オペランドで指定されるサブルーチンをCALLする (セグメント間)。

CSとPCレジスタの内容をスタックのトップにストアする, すなわち, CALLに続く命令のアドレスをスタックのトップにプッシュする. 後続の4個のメモリ・バイトの内容をPCとCSのレジスタに設定する. このバイトは次のように設定される.

1. この命令の2番目と3番目のバイトをPCレジスタにストアする.
2. この命令の4番目と5番目のバイトをCSレジスタにストアする.

命令コードを次に示す.



注)

1. CALLには次の4つのタイプがある.

CALL addr: セグメント間CALL

CALL mem: セグメント間インダイレクトCALL

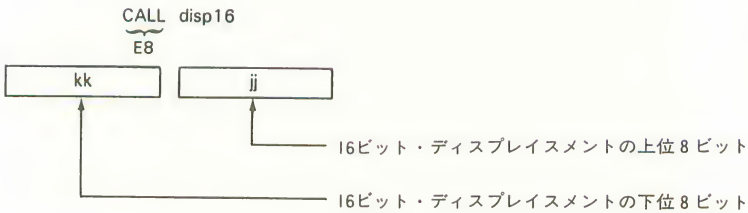
CALL disp: セグメント内CALL

CALL mem/reg: セグメント内インダイレクトCALL

CALL disp 16 (CALL)

オペランドで指定されるサブルーチンをCALLする (セグメント内).

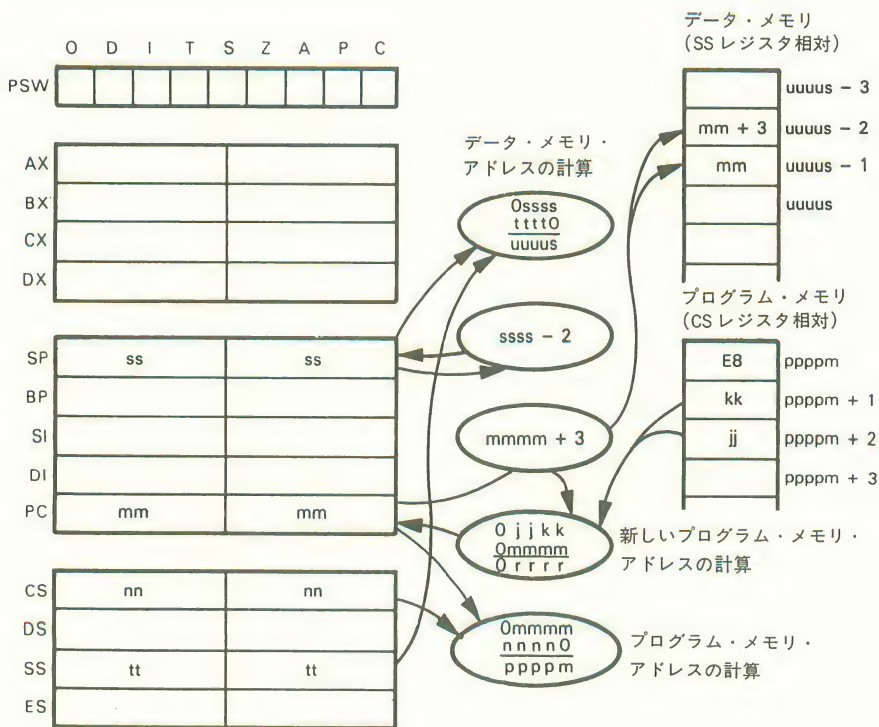
CALLに続く命令のアドレスをスタックのトップにプッシュする. 次の2つのプログラム・メモリ・バイトの内容を, 16ビットの符号なしディスプレイメントと見なして, プログラム・カウンタに加える. この位置から実行を続ける.



例として, 次の一連の命令を考える.

```
CALL SUBR
AND AL,7FH
-
-
-
SUBR PUSH AX
```

CALL命令実行後, AND命令のアドレスはスタックにプッシュされ, SUBRのPUSH命令が次に実行される.



注)

1. CALLには次の4つのタイプがある。

CALL disp: セグメント内CALL

CALL mem/reg: セグメント内インダイレクトCALL

CALL addr: セグメント間CALL

CALL mem: セグメント間インダイレクトCALL

CALL mem (CALL)

オペランドで指定されるサブルーチンをCALLする (セグメント間)。

CSとPCのレジスタの内容をスタックのトップにストアする。すなわち、CALLに続く命令のアドレスをスタックにプッシュする。指定されたメモリ位置のワードをPCレジスタに移し、それに続くワードをCSレジスタに移す。この位置から実行を続ける。

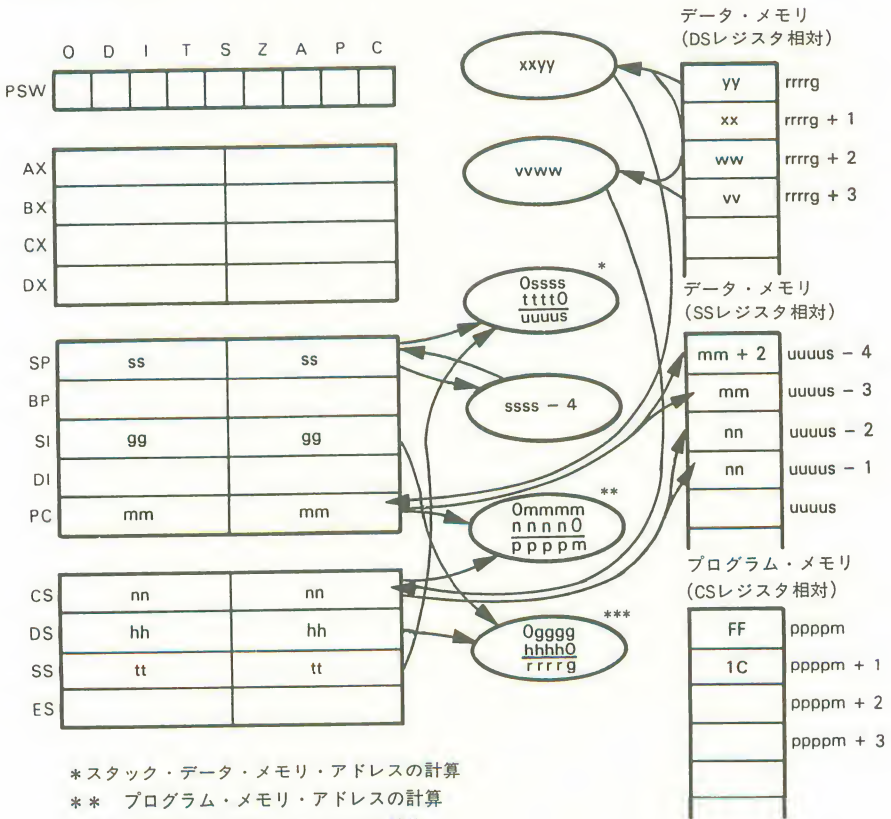
命令コードを次に示す。



D S レジスタが 0400_{16} を含み, S I レジスタが 0004_{16} を含み, 04004_{16} のメモリ・ワードが 0100_{16} で 04006_{16} のメモリ・ワードが $0FE0_{16}$ であると仮定する。

CALL [SI]

の実行後, P C レジスタは 0100_{16} を含み, C S レジスタは $0FE0_{16}$ を含む. 実行は $0FF00_{16}$ から続けられる。



CALL [SI]

サイクル数: 37 + EA

注)

1. CALLには次の4つのタイプがある.

CALL mem:セグメント間インダイレクトCALL

CALL addr:セグメント間CALL

CALL mem/reg:セグメント内インダイレクトCALL

CALL disp:セグメント内CALL

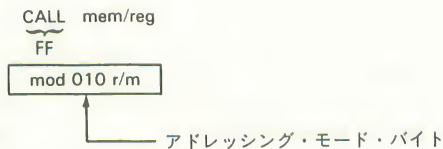
2. mod=11のとき, この操作は定義されない.

CALL mem/reg (CALL)

オペランドで指定されるサブルーチンをCALLする (セグメント内).

CALLに続く命令のアドレスをスタックのトップに格納する. 指定されたオペランドがレジスタならば, そのレジスタの内容をPCレジスタに移動する. 指定されたオペランドがメモリ位置ならば, 指定されたメモリ位置の内容をPCレジスタに移動する. この位置から実行を続ける.

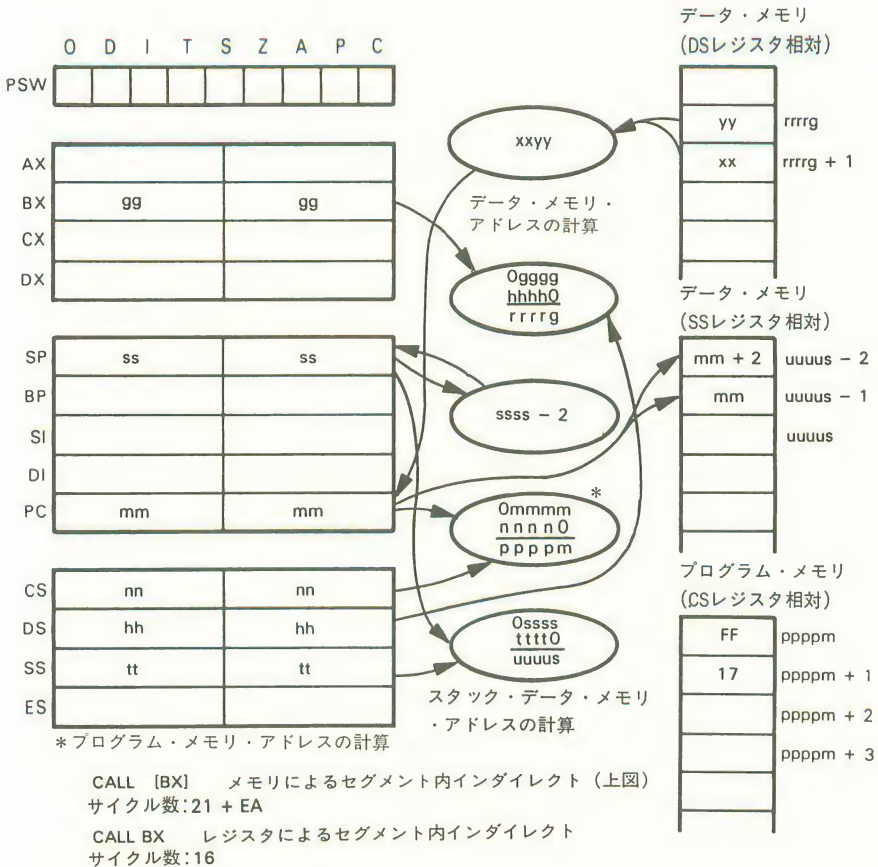
命令コードを次に示す.



PCレジスタが $FF00_{16}$ を含み, DSレジスタが 0100_{16} を含み, BXレジスタが 0026_{16} を含み, メモリ位置 01026_{16} のワードが 0240_{16} の場合を考える.

CALL [BX]

の実行後, PCレジスタは 0240_{16} を含む. この位置から実行が続けられる.



注)

1. `CALL` には次の4つのタイプがある。

- `CALL mem/reg`: セグメント内インダイレクト `CALL`
- `CALL disp`: セグメント内 `CALL`
- `CALL mem`: セグメント間インダイレクト `CALL`
- `CALL addr`: セグメント間 `CALL`

CBW (Convert Byte to Word)

ALレジスタの符号をAHレジスタに拡張する。

ALレジスタの最上位ビットが1ならば FF_{16} をAHレジスタにストアし、そうでなければ 00_{16} をAHレジスタにストアする。

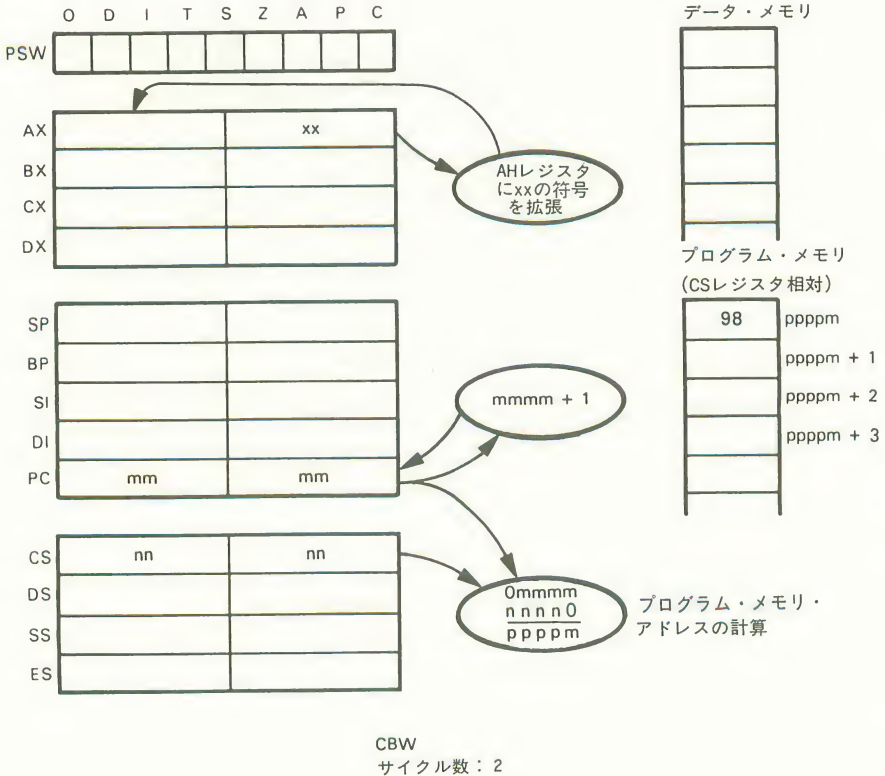
命令コードを次に示す。

CBW
98

例として、ALレジスタが $4F_{16}$ を含むならば、

CBW

の実行によって 00_{16} がAHレジスタにストアされる。



注)

1. ステータスは影響を受けない。
2. ALレジスタの値は、 $+127$ から -128 の間の数を表わす、すなわち、ALは符号付き8ビットの値を含んでいなければならない。
3. この命令は、16ビットのIMULあるいはIDIVの命令の前に、ALレジスタの拡張に使用できる。

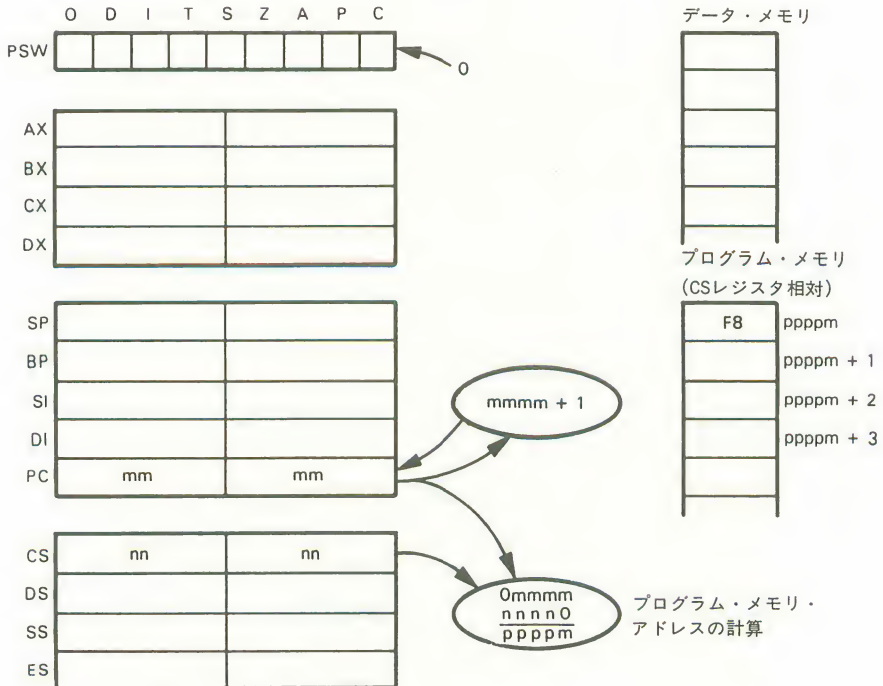
CLC (Clear Carry flag)

キャリー・フラグをクリアする。

この命令は、CFフラグを0に設定する。他のステータスあるいはレジスタは影響を受けない。

命令コードを次に示す。

CLC
F8



CLC
サイクル数: 2

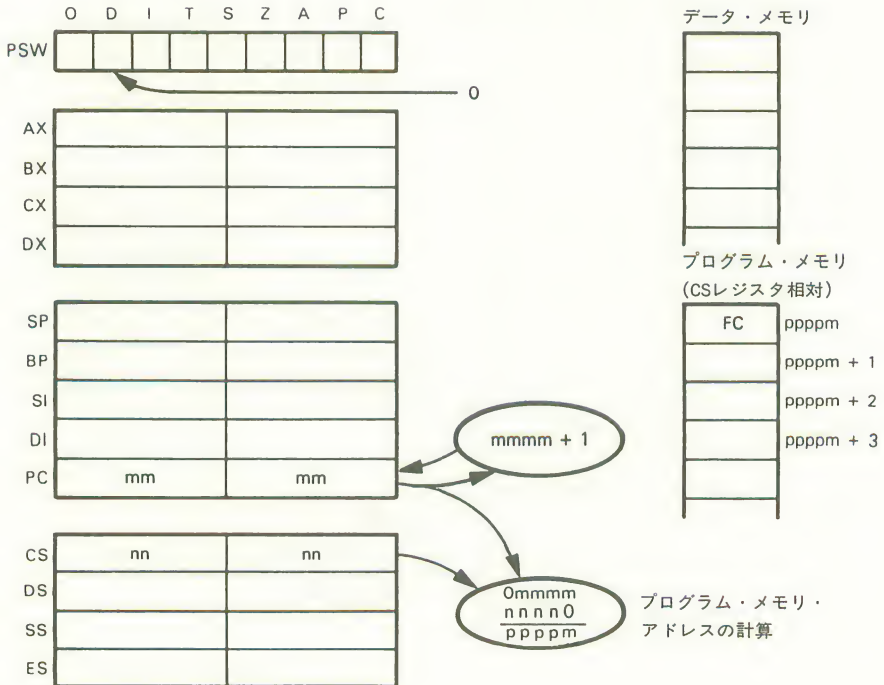
CLD (Clear Direction flag)

ディレクション・フラグをクリアする。

この命令は、DFフラグを0に設定する。これは、ストリング操作で用いられているポインタを、その操作において自動的に増加させる効果をもっている。他のステータスあるいはレジスタは影響を受けない。

命令コードを次に示す。

CLD
FC



CLD
サイクル数：2

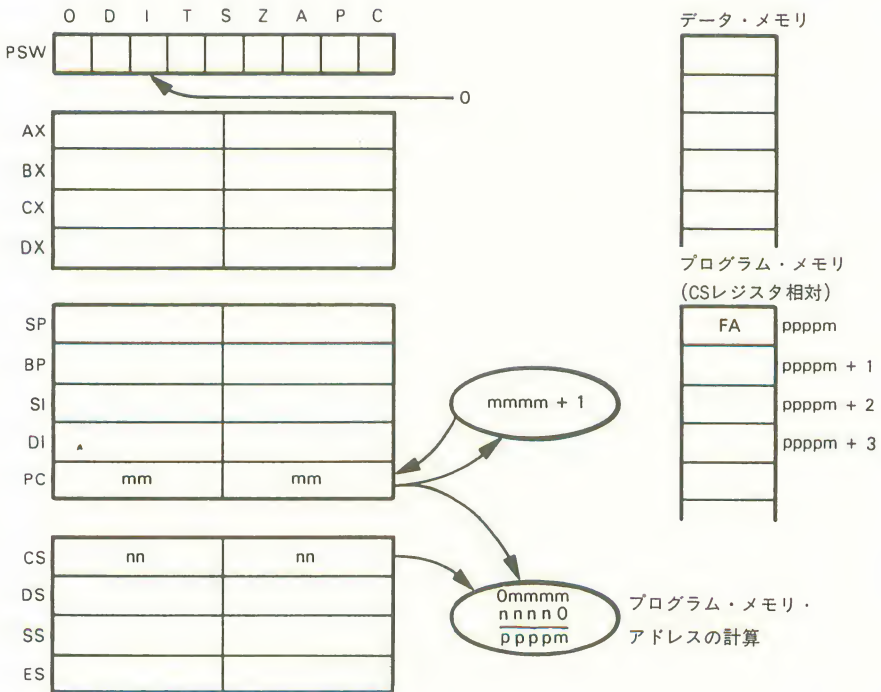
CLI (Clear Interrupt flag)

インタラプト・フラグをクリアする。

IFフラグを0に設定する。これは、NMIラインで生じるノンマスクابل・インタラプトを除くすべてのインタラプトを無効にする効果をもっている。

命令コードを次に示す。

CLI
FA



CLI
サイクル数: 2

注)

1. この命令は、8080の命令 DIと同じ機能を果たす。
2. 8086がインタラプト・リクエストを受け付けると、インタラプトは自動的に無効となることに注意。

CMC (Complement Carry flag)

キャリー・フラグのコンプリメントをとる。

キャリー・フラグのコンプリメント（反転）を行なう。他のステータスあるいはレジスタは影響を受けない。

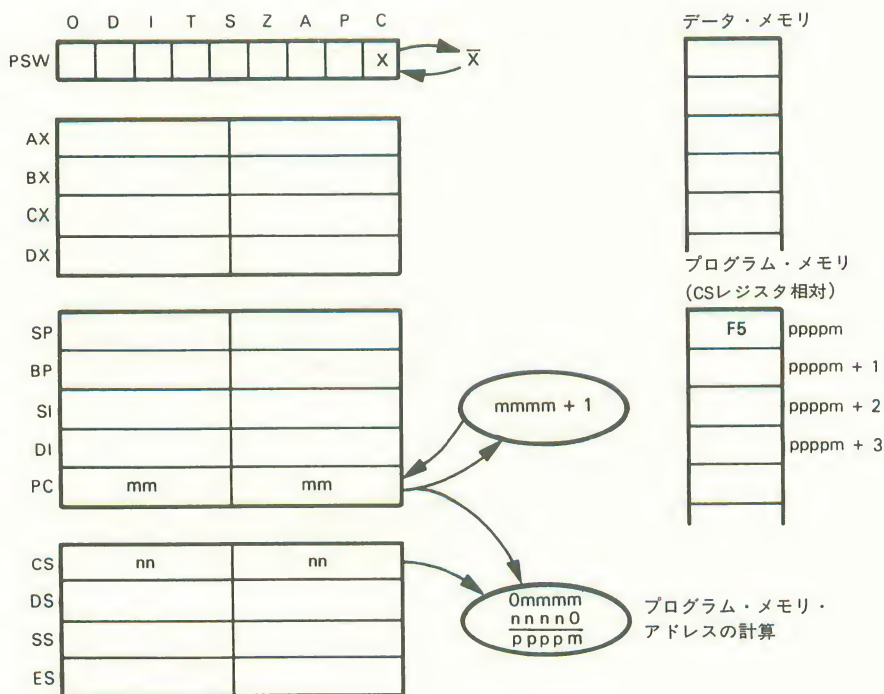
命令コードを次に示す。

CMC
F5

例えば、キャリー・フラグが0のとき、

CMC

を実行すると、キャリー・フラグは1となる。



CMC
サイクル数：2

注)

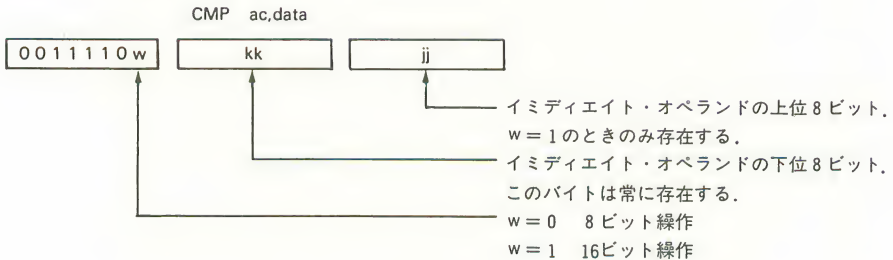
1, この命令は、8080の命令 CMC と同じ機能を果たす。

CMP ac, data (Compare)

アキュムレータとイミディエイト・データを比較する。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データと、ALレジスタ（8ビット操作）あるいはAXレジスタ（16ビット操作）とを比較するために用いられる。比較は、指定されたレジスタからイミディエイト・バイトのデータを減算し、その結果をフラグに設定することによって行なわれる。この操作の結果は指定されたレジスタにストアはされず、したがってレジスタは何の影響も受けず、ステータスだけが変化する。

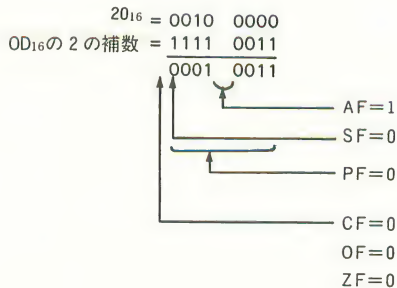
命令コードを次に示す。

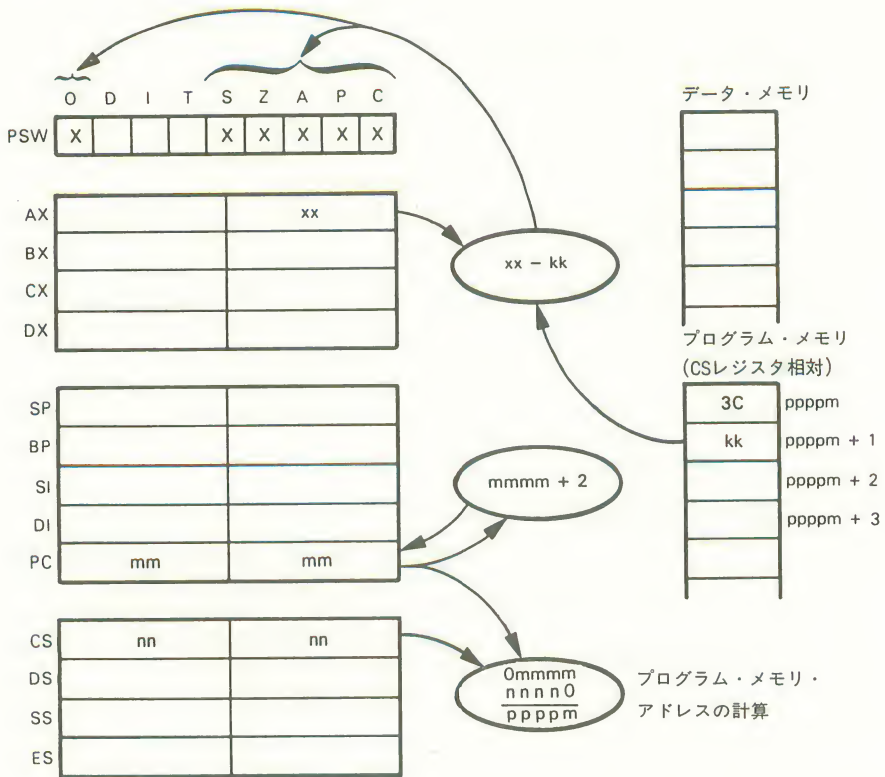


ALレジスタが 20_{16} を含む場合を考える。

CMP AL, 0DH

の実行後、ALレジスタの値は 20_{16} のままであるが、ステータスは次のように変更される。





注)

1. イミディエイト・データと、他の汎用レジスタあるいはメモリの内容との比較が必要な場合は、CMP mem/reg, data 命令を参照。
2. この命令は、8080の命令 C P I data と同じ機能を果たす。さらに、16ビットの比較も可能。

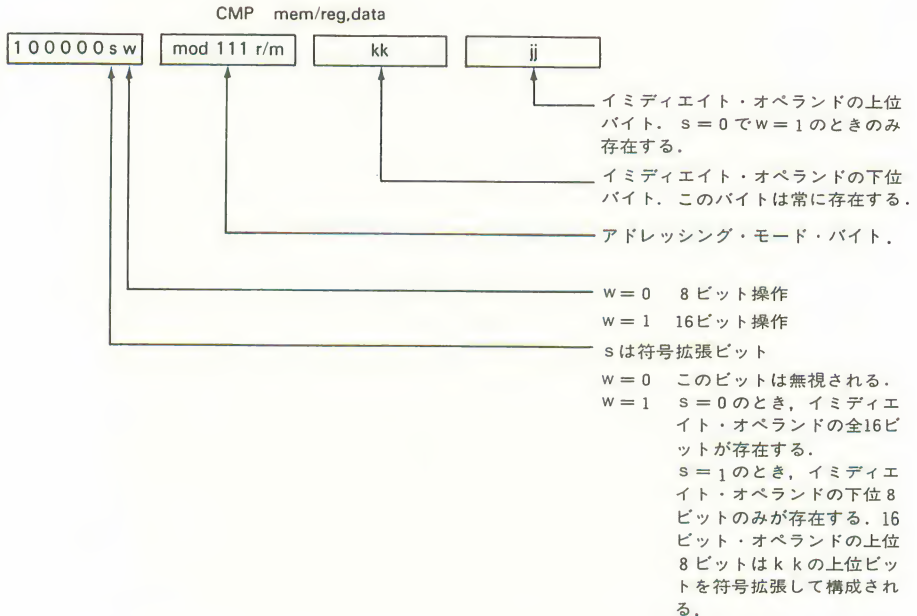
CMP mem/reg, data (Compare)

レジスタあるいはメモリとイミディエイト・データを比較する。

この命令は、後続のプログラム・メモリ・バイトに存在するイミディエイト・データと、指定されたレジスタあるいはメモリ位置との比較を行なう。比較は、指定されたメモリ位

置あるいはレジスタからイミディエイト・バイトのデータを減算し、その結果をフラグに設定することによって行なわれる。この操作の結果は指定されたレジスタあるいはメモリ位置には格納されず、したがって、レジスタあるいはメモリは何の影響も受けず、ステータスだけが変化する。8ビットあるいは16ビットの操作が指定できる。

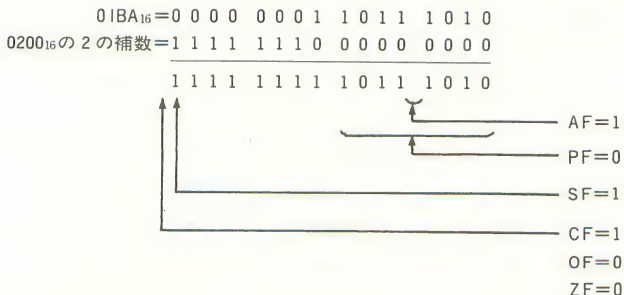
命令コードを次に示す。

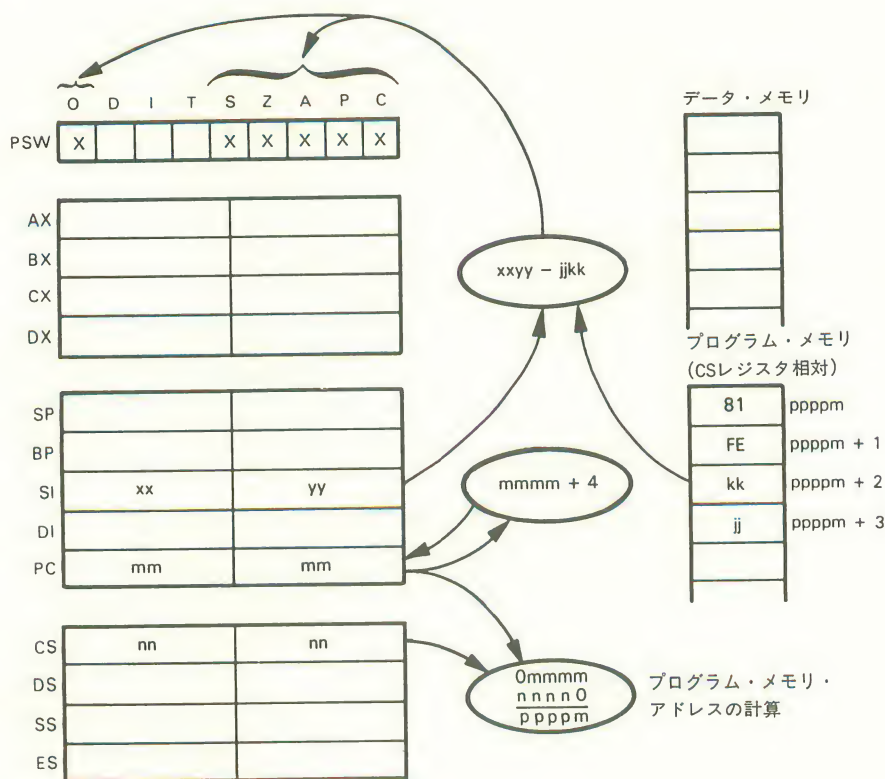


SIレジスタが01BA₁₆を含むと仮定する。

CMP SI,0200H

の実行後、SIレジスタの値は01BA₁₆のままであるが、ステータスは次のように変更される。





CMP SI, jkk

サイクル数: レジスタ・オペランド: 4

メモリ・オペランド: 10+EA

注)

1. この命令は、普通AXあるいはALのレジスタとイミディエイト・データを比較するためには用いられない。この目的のためには、命令 `CMP ac, data` がある。

CMP mem/reg₁, mem/reg₂ (Compare)

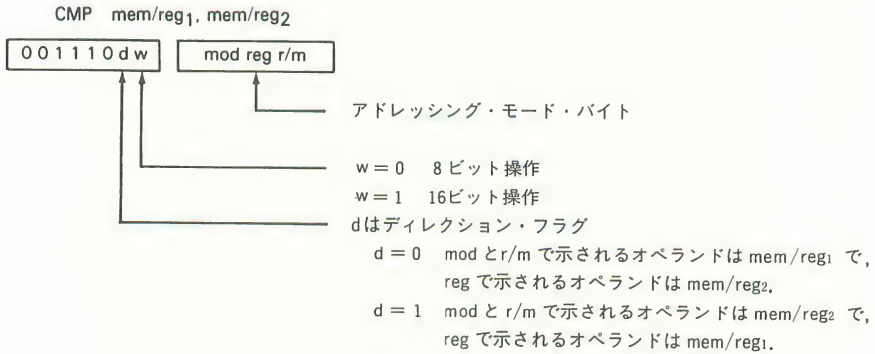
{ レジスタとレジスタ
 レジスタとメモリ
 メモリとレジスタ } の比較を行なう。

mem/reg₂で指定されるレジスタあるいはメモリのオペランドのデータと、mem/reg₁で指定されるレジスタあるいはメモリのオペランドのデータとの比較を行なう。比較はmem/reg₁で指定されるデータからmem/reg₂で指定されるデータを減算し、その結果をフラグ

に設定することによって行なわれる。

この操作によって、mem/reg₁あるいはmem/reg₂のどちらも影響を受けない。8あるいは16ビットの操作が指定できる。

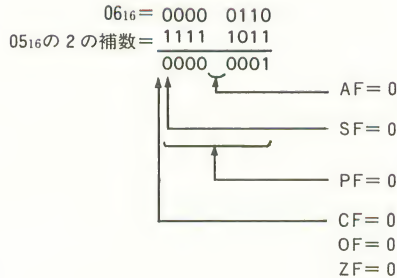
命令コードを次に示す。

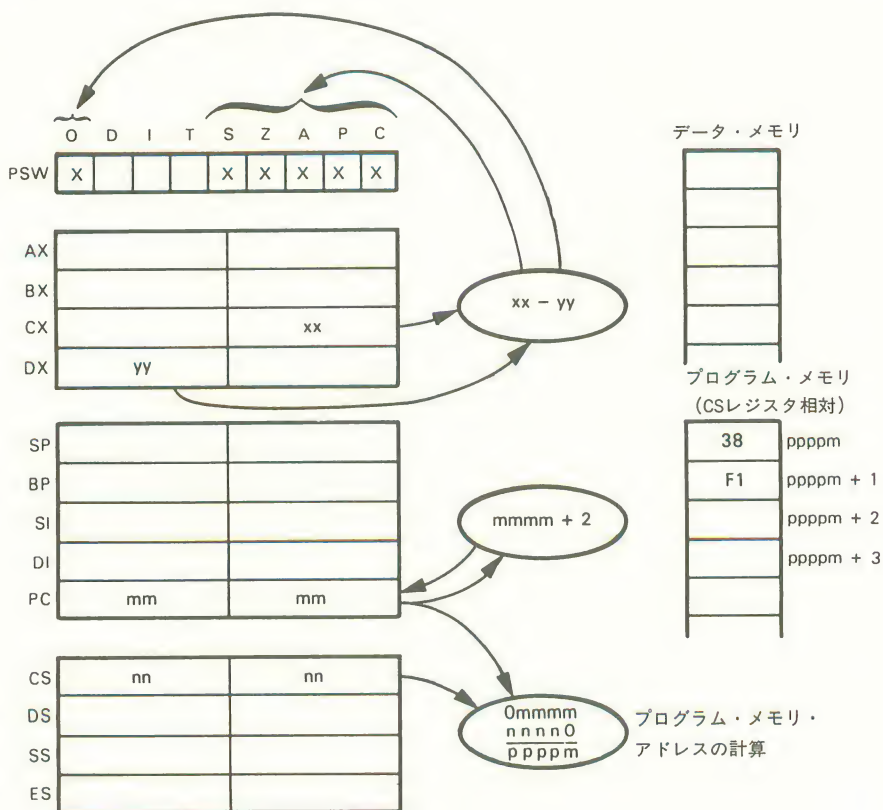


DHレジスタが05₁₆を含み、CLレジスタが06₁₆を含むと仮定する。

CMP CL,DH

の実行後、CLあるいはDHのレジスタのどちらも影響を受けないが、フラグは次のように設定される。





CMP CL, DH

サイクル数：レジスタとレジスタ：3

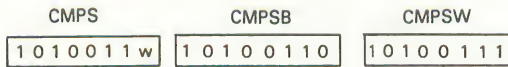
メモリとレジスタ：9 + EA

レジスタとメモリ：16 + EA

CMPS/CMPSB/CMPSW (Compare String)

メモリとメモリを比較する。

SIレジスタで示されるメモリ位置の内容とDIレジスタで示されるメモリ位置の内容の比較を行なう。比較は、SIレジスタで示されるメモリ位置の内容からDIレジスタで示されるメモリ位置の内容を減算し、その結果をフラグに設定することによって行なわれる。減算に用いられるメモリ位置のどちらも影響を受けない。SIとDIのレジスタは、DFフラグの値に応じて増加あるいは減少する。8あるいは16ビットの指定ができる。命令コードを次に示す。



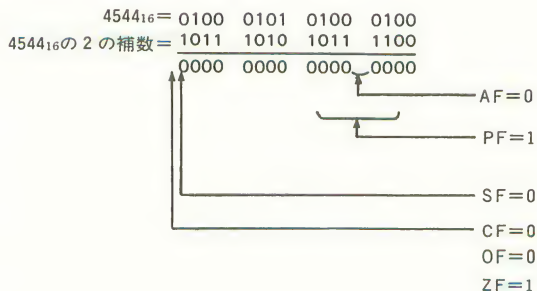
w = 0 8ビットの比較。DF = 0のとき、SIとDIのレジスタの値は1だけ増加する。DF = 1のとき、SIとDIのレジスタの値は1だけ減少する。

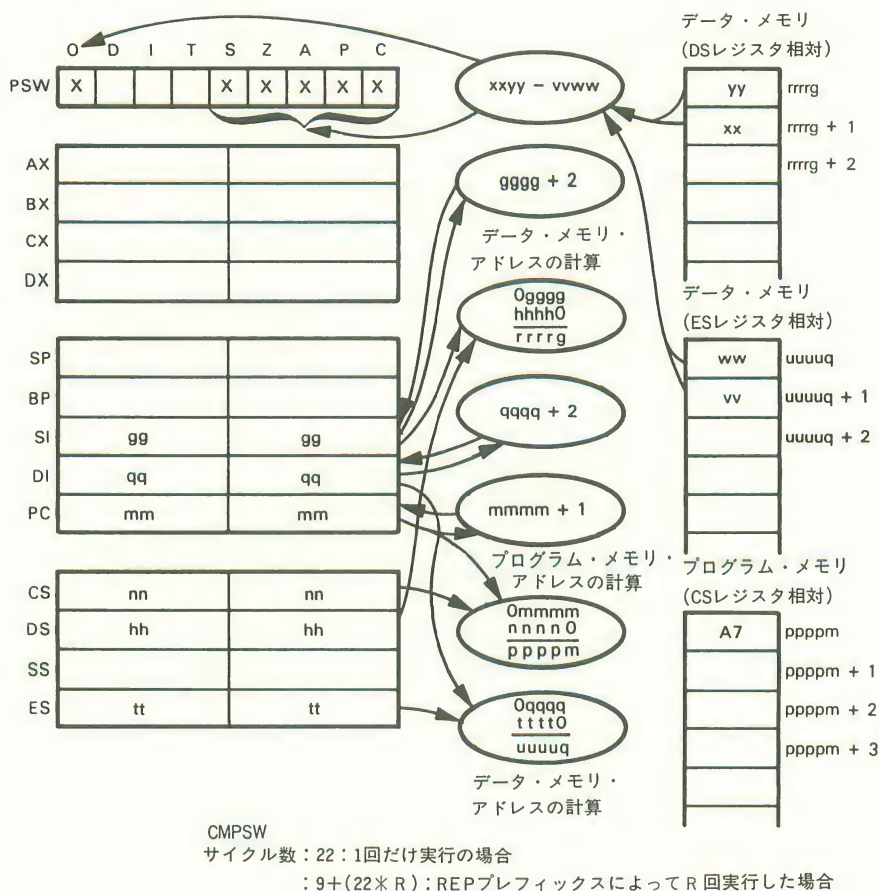
w = 1 16ビットの比較。DF = 0のとき、SIとDIのレジスタの値は2だけ増加する。DF = 1のとき、SIとDIの値は2だけ減少する。

DFフラグが0で、DSレジスタが0600₁₆を含み、SIレジスタが0108₁₆を含み、ESレジスタが0060₁₆を含み、DIレジスタが0188₁₆を含み、メモリ位置06108₁₆のワードが4544₁₆で、メモリ位置00788₁₆のワードが4544₁₆であると仮定する。

CMPSW

の実行後、SIレジスタは010A₁₆に、DIレジスタは018A₁₆となり、フラグは次のように設定される。





注)

- REPプレフィックスやLOCKプレフィックスはこの命令と共に用いられる。この命令と共に、LOCKプレフィックスとREPプレフィックスの両方が用いられた場合は問題となる。この件についての解説は第4章を参照のこと。
- SIレジスタによって示されるオペランドのデフォルト・セグメント・レジスタはDSである。このセグメント・レジスタは、セグメント変更プレフィックスを用いて変えられる。DIによって示されるオペランドのデフォルト・セグメント・レジスタはESである。このセグメント・レジスタの指定は無効にならない。
- インテルのアセンブラは、バイトとワードの形式に加えて、汎用の形式を許している。汎用の形式に対して、アセンブラは、8ビットあるいは16ビットのどちらの比較が行なわれるのかを決められる情報を必要とする。これがどのように行なわれるかについての説明は、この章の最後を参照のこと。

4. REPプレフィックスを伴ったCMP Sの実行時間は、次のように表わせる。

$$\underbrace{\text{REP}}_{2+} \quad \underbrace{\text{CMPS}}_{9+(22 \times R)}$$

もし $R=10$ ならば、実行時間は231クロック・サイクルとなる。

CWD (Convert Word to Doubleword)

A Xレジスタの符号をD Xレジスタに拡張する。

A Xレジスタの最上位ビットが1ならば、 FFFF_{16} をD Xレジスタにストアし、そうでなければ 0000_{16} をD Xレジスタにストアする。

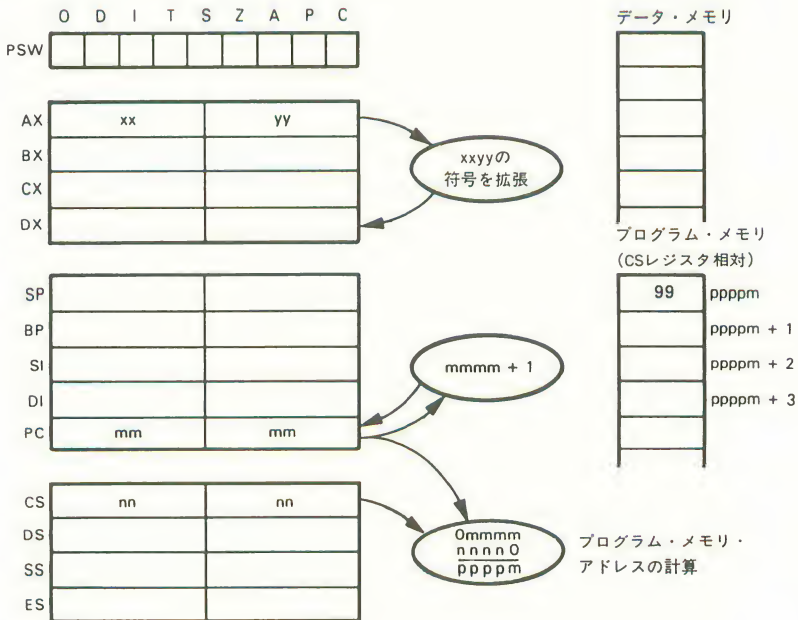
命令コードを次に示す。

CWD
99

A Xレジスタが B001_{16} を含むと仮定する。

CWD

の実行後、D Xレジスタは FFFF_{16} を含む。



CWD

サイクル数: 5

注)

1. ステータスは影響を受けない。
2. この命令は除算を行なうときに有用である。16ビットの除数が用いられるならば、32ビットの被除数が必要となる。AXレジスタにのみ有効なビットが存在するならば、この命令は32ビットの被除数を得るために符号ビットをDXレジスタに拡張する。この方法はIDIV命令に最適であることに注意。

DAA (Decimal Adjust for Addition)

加算後のアキュムレータの内容を10進数に変換する。

ALレジスタの内容を、2進化10進(BCD)の形式に変換する。この命令は、2つのBCDの加算後にだけ用いられる。すなわち、ADD DAAあるいはADC DAAを、BCDの結果を得るためにBCDのソース・オペランドを操作する複合の10進演算命令と見なす。

変換のアルゴリズムを次に示す。

1. AFフラグが1、あるいはALレジスタの下位4ビットがAからFまでの値ならば、ALレジスタに 06_{16} を加算し、AFフラグを1にする。
2. CFフラグが1、あるいはALレジスタの上位4ビットが9よりも大きければ、ALレジスタに 60_{16} を加算し、CFフラグを1にする。

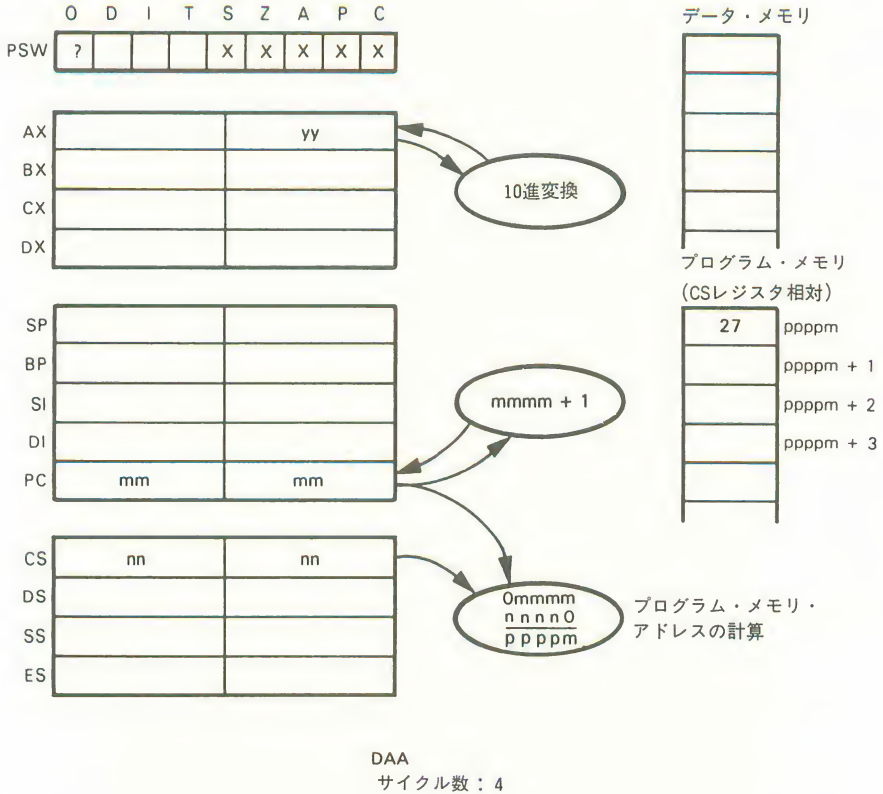
命令コードを次に示す。

$$\begin{array}{c} \text{DAA} \\ \hline 27 \end{array}$$

ALレジスタが 28_{16} を含み、BLレジスタが 68_{16} を含むと仮定する。

```
ADD AL, BL
DAA
```

の実行後、ALレジスタは、 90_{16} ではなくて 96_{16} となる。



注)

1. この命令は、2つのパック形式BCDオペランドの加算に有用である。2つのパック形式オペランドの減算の変換については、DAS命令を参照。ASCIIの加算と減算の結果の変換については、AAAとAASの命令を参照。

DAS (Decimal Adjust for Subtraction)

減算後のアキュムレータの内容を10進数に変換する。

この命令は、ALレジスタの内容を2進10進(BCD)の形式に変換する。また、この命令は、2つのBCDの減算後にだけ用いられる。すなわち、SUB DASあるいはSBB DASを、BCDの結果を得るためにBCDのソース・オペランドを操作する複合の10進演算命令と見なす。

変換のアルゴリズムを次に示す。

1. AFフラグが1、あるいはALレジスタの下位4ビットがAからFの間ならば、ALレジスタから06₁₆を減算し、AFフラグを1にする。

注)

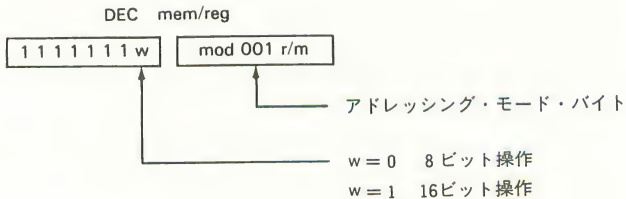
1. これは2つのパック形式BCDに対する10進減算のアジャストメント・アルゴリズムである。減算のアジャストメントに有効な他の操作にAAS命令があり、これはASCIIの減算結果のアジャストを行なう。

DEC mem/reg (Decrement)

レジスタあるいはメモリの内容をデクリメントする。

指定されたレジスタあるいはメモリ位置の内容から1を減じる。8あるいは16ビットの操作が指定できる。

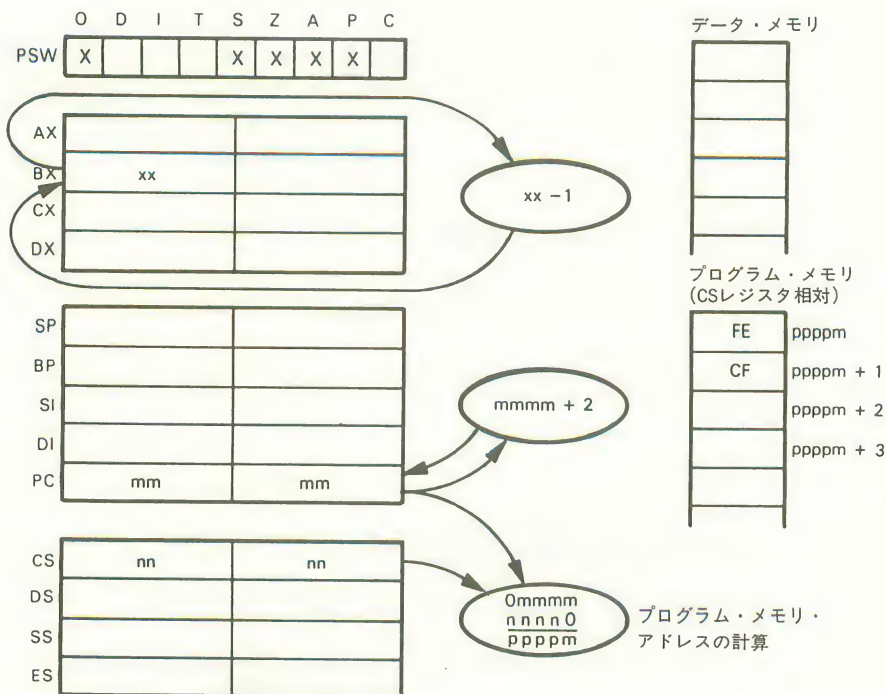
命令コードを次に示す。



BHレジスタが $4F_{16}$ を含むと仮定する。

DEC BH

の実行後、BHレジスタは $4E_{16}$ となる。



DEC BH
 サイクル数：レジスタ・オペランド：3
 メモリ・オペランド：15+EA

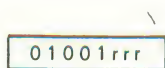
注)

1. この命令は、8080の命令 DCR reg と同じ機能を果たす。種々のアドレッシング・モードが利用可能で8あるいは16ビットが選択できることから、この命令は8080の命令と比較して非常に有力であることに注意。
2. この命令でセグメント・レジスタはデクリメントできない。
3. この命令は、通常16ビット・レジスタをデクリメントするためには用いられない。命令 DEC reg はこの機能を果たし、しかもプログラム・メモリが1バイトですむ。この命令は、8ビット・レジスタあるいはメモリをデクリメントするために用いられる。
4. この命令は、キャリー・フラグに影響を与えない。

DEC reg (Decrement)

レジスタをデクリメントする。

指定されたレジスタの内容から1を減じる。これは16ビットのデクリメント命令である。命令コードを次に示す。



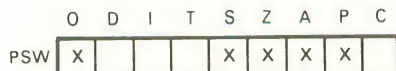
対象となる16ビットレジスタを3ビットで指定

$rrr=000$: AX
 001: CX
 010: DX
 011: BX
 100: SP
 101: BP
 110: SI
 111: DI

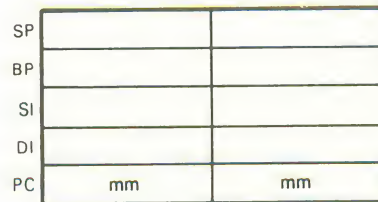
例として、CXレジスタが0200₁₆を含む場合を考える。

DEC CX

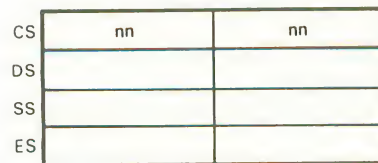
の実行結果、CXレジスタの内容は01FF₁₆に減少している。



$xyxy - 1$



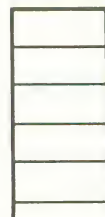
$mmmm + 1$



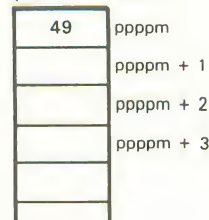
$$\begin{array}{r} 0mmmm \\ nnnn0 \\ ppppm \end{array}$$

プログラム・メモリ・アドレスの計算

データ・メモリ



プログラム・メモリ
(CSレジスタ相対)



DEC CX
 サイクル数: 2

注)

1. この命令は、8080の命令 DCX reg と同じ機能を果たす。
2. セグメント・レジスタは、この命令を用いてデクリメントされない。

DIV mem/reg (Divide)

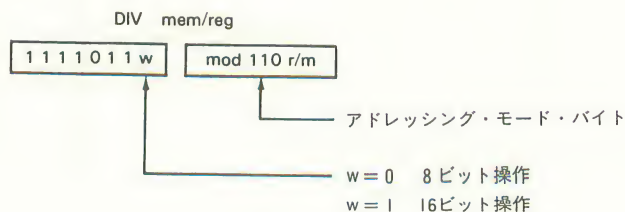
AH:ALあるいはDX:AXを、レジスタあるいはメモリの内容で割る。

両方のオペランドを符号なし2進級と見なして、AH:AL(8ビット操作)あるいはDX:AX(16ビット操作)のレジスタを、指定されたレジスタあるいはメモリ位置の内容で割る。16ビットの符号なし数値を8ビットの符号なし数値で割ること、あるいは32ビットの符号なし数値を16ビットの符号なし数値で割ることが可能である。8ビット操作の場合は、8ビットの商はALレジスタに返され、8ビットの余りはAHレジスタに返される。もしALレジスタに返される商が FF_{16} より大きいと、タイプ0(0による除算)のインタラプトが発生する。16ビットの操作では、16ビットの商はAXレジスタに返され、16ビットの余りはDXレジスタに返される。もしAXレジスタに返される商が $FFFF_{16}$ より大きいと、タイプ0(0による除算)のインタラプトが発生する。

0による除算のインタラプトの結果、次の動作が行なわれる。

1. フラグ・レジスタをスタックにプッシュする。
2. IFとTFのフラグをクリアする。
3. CSレジスタをスタックにプッシュする。
4. メモリ位置 00002_{16} のワードをCSレジスタにロードする。
5. PCをスタックにプッシュする。
6. メモリ位置 00000_{16} のワードをPCレジスタにロードする。

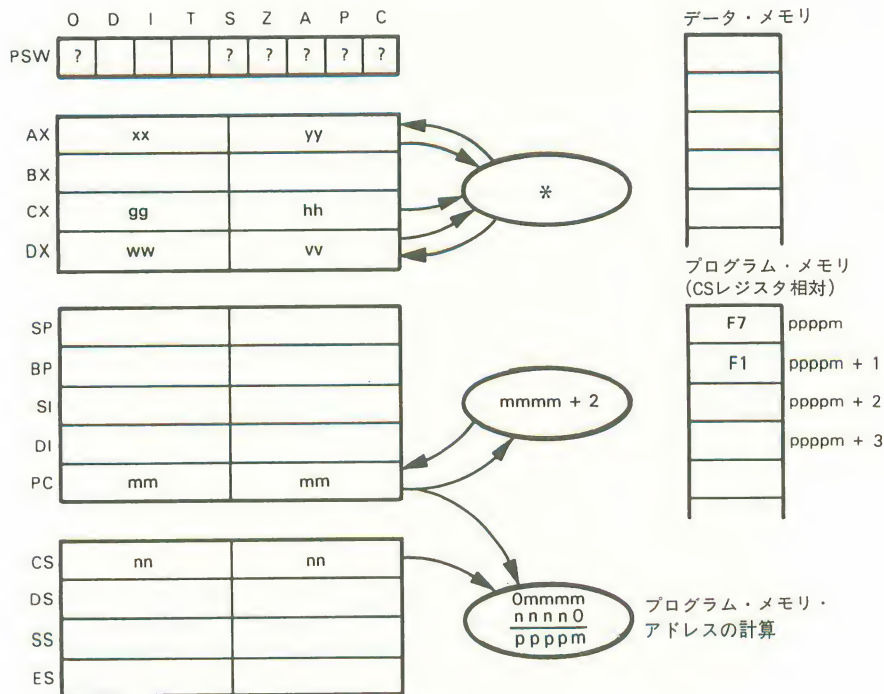
命令コードを次に示す。



例として、AXレジスタが $0F05_{16}$ を含み、DXレジスタが $068A_{16}$ を含み、CXレジスタが $08E9_{16}$ を含む場合を考える。

DIV CX

の実行後、AXレジスタは商 $BBE1_{16}$ 、DXレジスタは余り $073C_{16}$ となる。OF, SF, ZF, AF, PF, CFのフラグの値は、この操作で不定となる。すなわち、DIV命令後の特定のフラグがとる値は分からない。



* wwwvxyy が gg hh によって割られる。

商は A X に戻され、余りは D X に戻される。

DIV CX

サイクル数：16ビットのメモリによる除算：(150-168)+EA

8ビットのメモリによる除算：(86-96)+EA

16ビットのレジスタによる除算：144-162

8ビットのレジスタによる除算：80-90

注)

- この命令の実行後、算術演算のフラグの値は不定となる。
- DIV命令の実行以前に、DIV命令が結果として0による除算のインタラプトを発生するかを知る必要があるならば、次に示す一連の命令は有用である。

16ビット除算：CLが除数を含むと仮定。

```
CMP AH, CL
JNB OVERFLOW$HANDLER
```

32ビット除算：BXが除数を含むと仮定。

```
CMP DX, BX
JNB OVERFLOW$HANDLER
```

この種類のチェックは、0による除算のインタラプト処理が十分でない場合に有用と

なる。

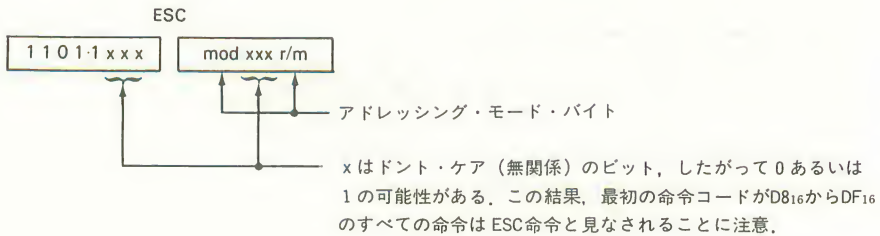
3. 被除数と除数が同一の長さのときは、最初に被除数はCBWあるいはCWDの命令を用いて16ビットあるいは32ビットに拡張しておかなければならない。

ESC mem (Escape)

メモリにアクセスする。

この命令は、指定されたメモリ位置の内容を、データ・バスに設定する。本質的には、8086に関する限りこの命令は何の操作も行わない。この命令は、他のプロセッサに8086のアドレッシング・モードを利用し、8086の命令の流れから命令を受け取らせるために用いられる。

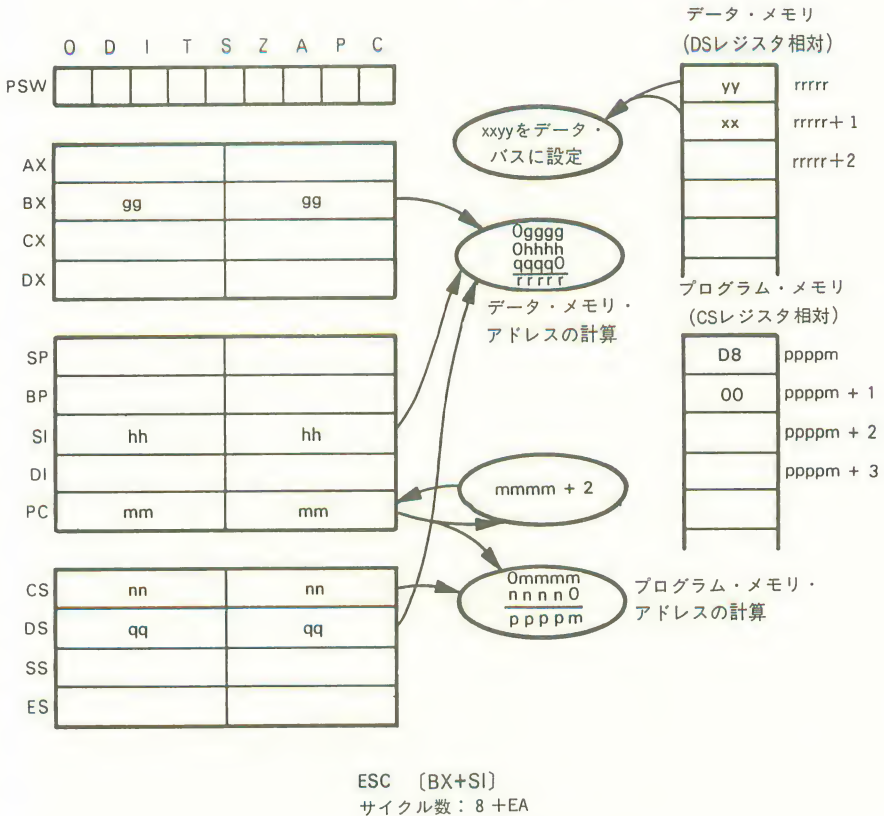
命令コードを次に示す。



BXレジスタが $063A_{16}$ を含み、SIレジスタが 0003_{16} を含み、DSレジスタが $FF80_{16}$ を含み、メモリ位置 $FFE3D_{16}$ のワードが $C308_{16}$ であるとする。

ESC [BX+SI]

を実行すると、指定されたメモリ素子によってREADYラインが確立された時点で、データ・ライン上には $C308_{16}$ のデータが存在する。



注)

1. $\text{mod}=11$ (すなわち、レジスタを指定)の場合は、この命令は無操作となり、CLOCK CYCLES = 2である。

HLT (Halt)

プロセッサをホルト状態にする。

HLT命令が実行されると、プログラムの実行は停止する。実行を再開するには、外部インタラプトあるいはリセットが必要となる。レジスタあるいはステータスは影響を受けない。

注意: HLT命令以前にSTI命令によってインタラプトが許可されていなければ、8086 CPUは、ハードウェアのリセットあるいはノンマスクابل・インタラプトによる場合を除いて、ホルト状態から抜け出すことができない。

命令コードを次に示す。

HLT
F4

	O	D	I	T	S	Z	A	P	C
PSW									

AX		
BX		
CX		
DX		

SP		
BP		
SI		
DI		
PC	mm	mm

CS	nn	nn
DS		
SS		
ES		

データ・メモリ

プログラム・メモリ
(CSレジスタ相対)

F4	ppppm
	ppppm + 1
	ppppm + 2
	ppppm + 3

mmmm + 1

0mmmm
nnnn0
ppppm

プログラム・メモリ・
アドレスの計算

HLT
サイクル数：2

IDIV mem/reg (Integer Divide)

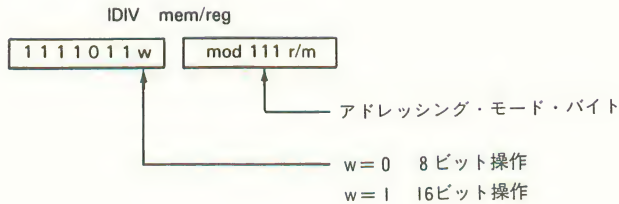
AH:ALあるいはDX:AXを、レジスタあるいはメモリの内容で割る。

両方のオペランドを符号付き2進数と見なして、AH:AL（8ビット操作）あるいはDX:AX（16ビット操作）のレジスタを、指定されたレジスタあるいはメモリ位置の内容で割る。16ビットの符号付き数値を8ビットの符号付き数値で割ること、あるいは32ビットの符号付き数値を16ビットの符号付き数値で割ることが可能である。8ビット操作の場合、8ビットの商はALレジスタに返され、8ビットの余りはAHレジスタに返される。返される商が $7F_{16}$ より大きければ、タイプ0（0による除算）のインタラプトが発生する。16ビット操作の場合、16ビットの商はAXレジスタに返され、16ビットの余りはDXレジスタに返される。AXレジスタに返される商が $7FFF_{16}$ より大きければ、タイプ0（0による除算）のインタラプトが発生する。

0による除算のインタラプトの結果、次の動作が行なわれる。

1. フラグ・レジスタをスタックにプッシュする。
2. IFとTFのフラグをクリアする。
3. CSレジスタをスタックにプッシュする。
4. メモリ位置 00002_{16} のワードをCSレジスタにロードする。
5. PCをスタックにプッシュする。
6. メモリ位置 00000_{16} のワードをPCレジスタにロードする。

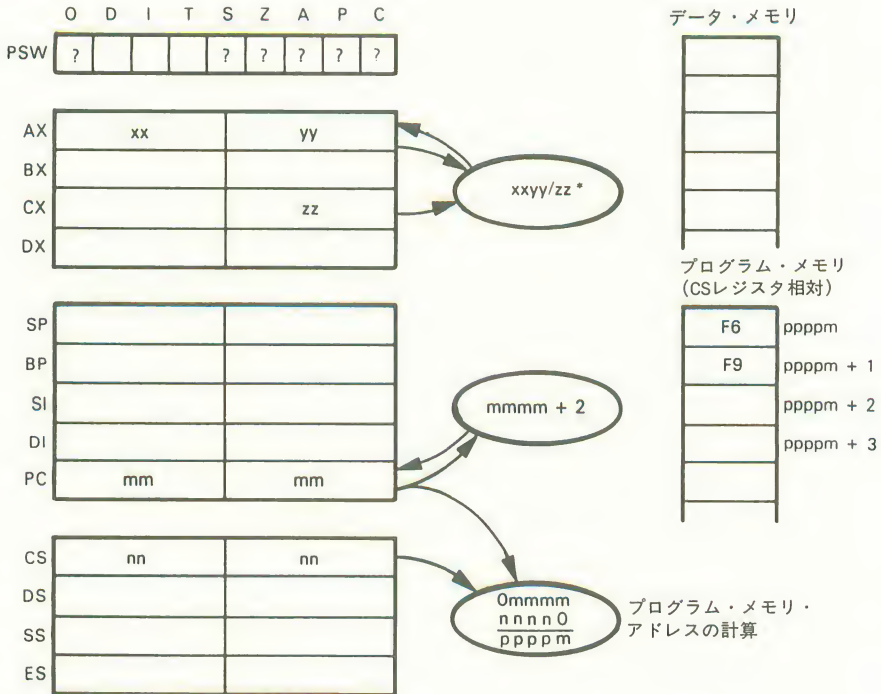
命令コードを次に示す。



CLレジスタが $0D_{16}$ を含み、AXレジスタが $00A9_{16}$ を含むと仮定する。

IDIV CL

の実行後、AXレジスタは $000D_{16}$ となる。



* 商は A レジスタに戻し、余りは A H レジスタに戻す。

IDIV CL

サイクル数：8ビットのメモリによる除算：(107—118)+EA

16ビットのメモリによる除算：(171—190)+EA

8ビットのレジスタによる除算：101—112

16ビットのレジスタによる除算：165—184

注)

1. これは、符号付き数値の除算命令である。両方のオペランドは、次の範囲の符号付き2進数として取り扱われる。

8ビット操作：+127～-128

16ビット操作：+32767～-32768

符号なしの除算については、DIV命令を参照。

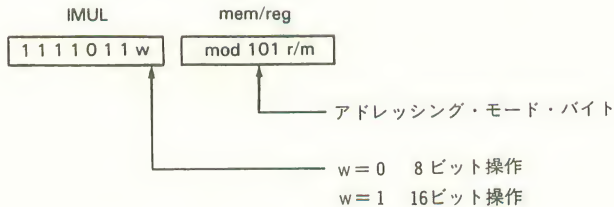
2. この命令の実行後、フラグの値は不定となる。
3. 被除数と除数が同一の長さのときは、最初に被除数は、CBWあるいはCWDの命令を用いて、16あるいは32ビットの形に変換されなければならない。

IMUL mem/reg (Integer Multiply)

ALあるいはAXのレジスタに、レジスタあるいはメモリの内容乗じる。

指定されたレジスタあるいはメモリ位置の内容と、AL（8ビット操作）あるいはAX（16ビット操作）とを、両方のオペランドを符号付き2進数と見なして乗じる。すなわち、符号付き乗算を行なう。8ビット操作の場合、結果の下位8ビットはALレジスタにストアされ、結果の上位8ビットはAHレジスタにストアされる。16ビット操作では、結果の下位16ビットはAXレジスタにストアされ、結果の上位16ビットはDXレジスタにストアされる。どちらの場合においても、結果の上位1/2が下位1/2の符号拡張ならばオーバーフローとキャリーのフラグは0となり、それ以外では1となる。（例えば、8ビット操作が行なわれて、AHレジスタに返される値が 00_{16} あるいは FF_{16} でなければ、キャリーとオーバーフローのフラグは1になる。）これらのフラグの値が1ならば、AHあるいはDXは有効な桁を含んでいることを意味する。

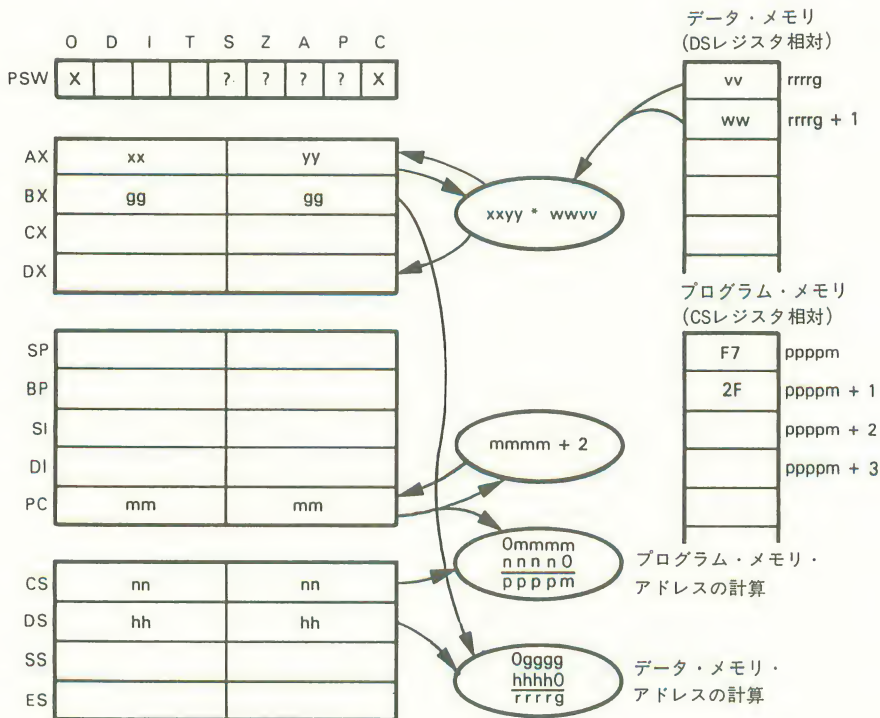
命令コードを次に示す。



例として、AXレジスタが $04E8_{16}$ を含み、DSレジスタが 0100_{16} を含み、BXレジスタが 0006_{16} を含み、メモリ位置 01006_{16} のワードが $4E20_{16}$ の場合を考える。

IMUL AX,[BX]

の実行後、AXレジスタは $4D00_{16}$ に、DXレジスタは $017F_{16}$ となり、キャリーとオーバーフローのフラグは1になる。



IMUL AX,[BX]

サイクル数: 8ビットのメモリによる乗算: (86-104)+EA

16ビットのメモリによる乗算: (34-160)+EA

8ビットのレジスタによる乗算: 80-98

16ビットのレジスタによる乗算: 128-154

注)

- これは符号付き数値の乗算操作である。両方のオペランドは次の範囲の数値として取り扱われる。

8ビット操作: +127~-128

16ビット操作: +32767~-32768

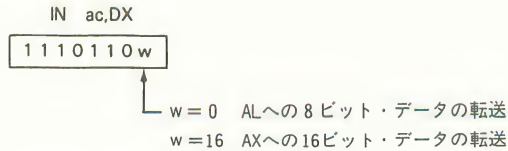
符号なしの乗算操作については、命令MULを参照。

- ある場合には、乗算を行なうためにシフトを用いることがより適当となるときがある。このような場合は、メモリの保存が最も重要ではなく速さが必要なときに生じる。
- この命令の実行後、SF、ZF、AF、PFのフラグは不定となる。

IN ac, DX (Input)

アキュムレータへの入力を行なう。

この命令は、DXレジスタの内容で指定されるI/Oポートから、AL(8ビット転送)あるいはAX(16ビット転送)のレジスタへ、8あるいは16ビットのデータ要素をロードする。命令コードを次に示す。

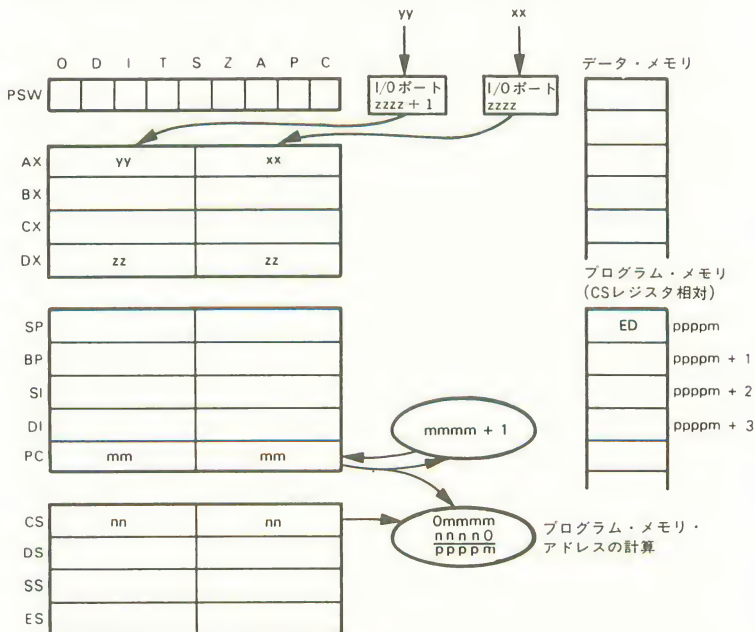


他のレジスタ(ALあるいはAXを除く)あるいはステータスは、影響を受けない。

DXレジスタが 1234_{16} を含み、ポート 1234_{16} のI/Oバッファが 23_{16} を含み、ポート 1235_{16} のI/Oバッファが $F4_{16}$ を含むと仮定する。

IN AX, DX

を実行すると、ALレジスタに 23_{16} が、AHレジスタに $F4_{16}$ がロードされる。



IN AX, DX
サイクル数: 8

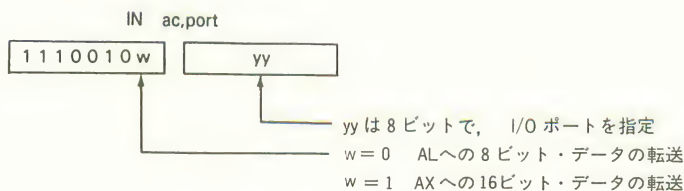
注)

1. この命令で、0 から FFFF_{16} までのアドレスを割り当てられている入力ポートにアクセスすることができる。
2. ac として、8 ビットに対しては AL が、16 ビットに対しては AX のみが指定できる。

IN ac, port (Input)

アキュムレータへの入力を行なう。

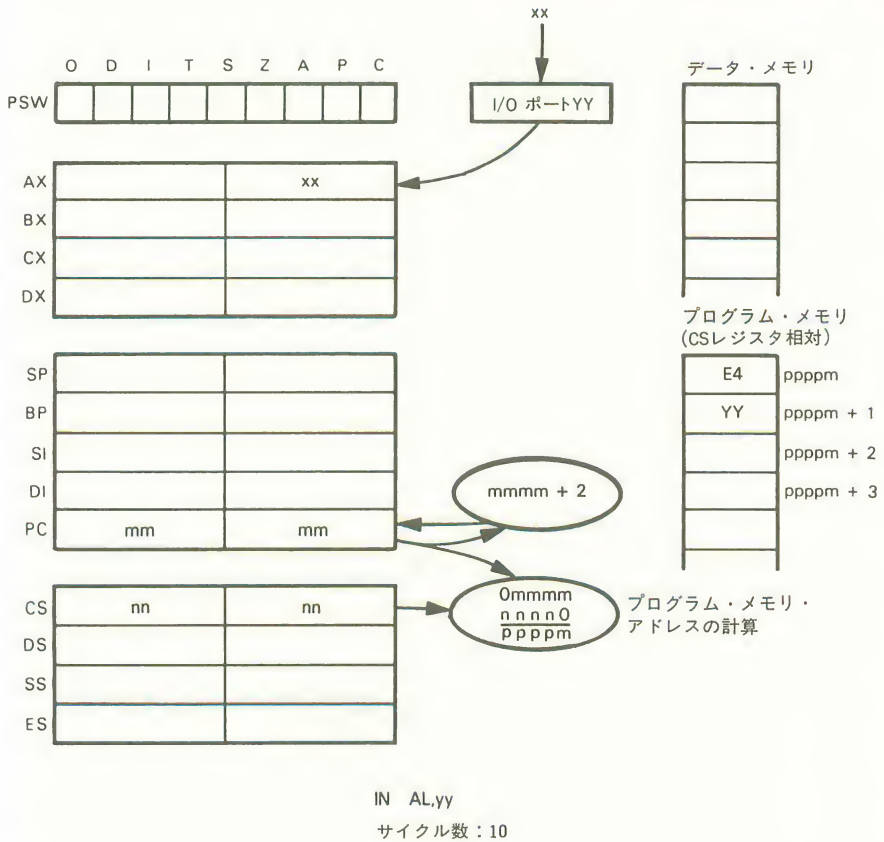
この命令は、命令の2番目のバイトで指定される I/O ポートから AL (8 ビット転送) あるいは AX (16 ビット転送) のレジスタへ 8 あるいは 16 ビットのデータ要素をロードする。命令コードを次に示す。



他のレジスタ (AL あるいは AX を除く) あるいはステータスは影響を受けない。ポート 06_{16} の I/O バッファが 43_{16} を含むと仮定する。

IN $\text{AL}, 06\text{H}$

の実行によって、 AL レジスタに 43_{16} がロードされる。



注)

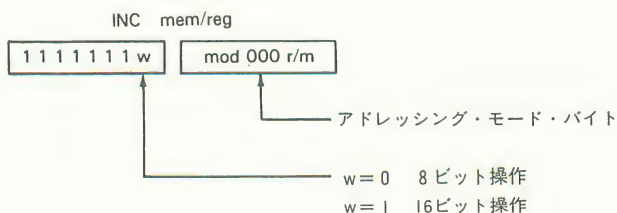
1. この命令で、0 から FF_{16} までのアドレスを割り当てられている I/O ポートにアクセスすることができる。この範囲外のアドレスのポート指定は、命令 `IN ac, DX` を参照。
2. この命令は、8080 の命令 `IN port` と同じ機能を果たす。
3. `ac` として、8 ビットに対しては `AL` が、16 ビットに対しては `AX` のみが指定できる。

INC mem/reg (Increment)

レジスタあるいはメモリの内容をインクリメントする。

指定されたレジスタあるいはメモリ位置の内容に 1 を加える。8 あるいは 16 ビットの操作が指定できる。

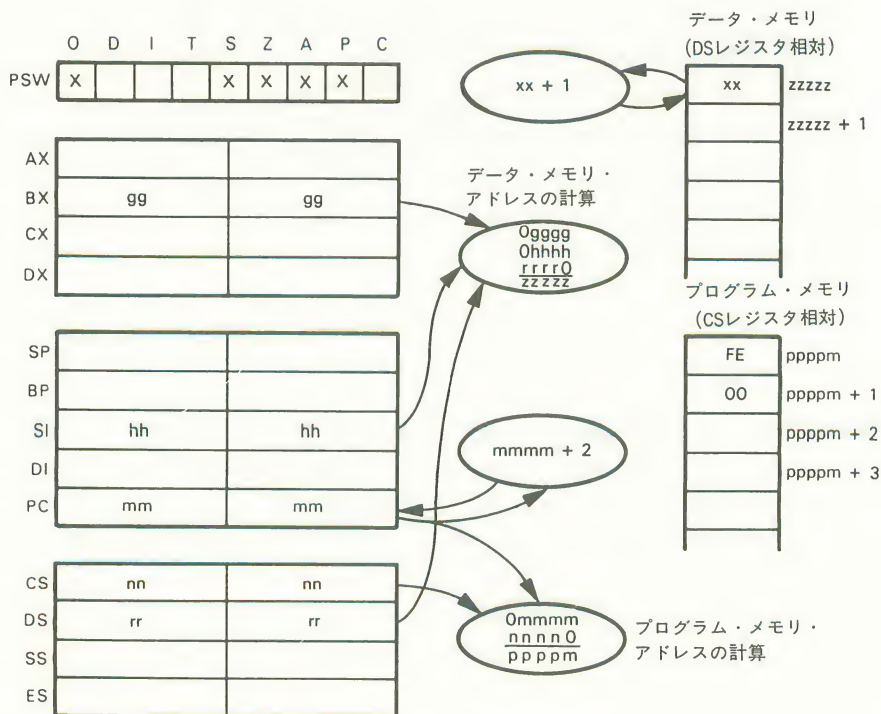
命令コードを次に示す。



D S レジスタが $F800_{16}$ を含み, B X レジスタの内容が 0280_{16} で, S I レジスタが $001E_{16}$ を含み, メモリ位置 $F829E_{16}$ が 64_{16} の場合を考える.

INC [BX+SI]

の実行後, メモリ位置 $F829E_{16}$ は 65_{16} を含む.



INC [BX + SI]

サイクル数: メモリ・オペランド: 15+EA

レジスタ・オペランド: 3

注)

1. セグメント・レジスタはこの命令でインクリメントはできない。
2. この命令は、8080の命令 `INR reg` と同じ機能を果たす。またこの命令は、8080の命令と比較して非常に有力であることに注意。
3. この命令は、通常16ビットのレジスタをインクリメントするためには用いられない。命令 `INC reg` はこの機能を果たし、しかもプログラム・メモリが1バイトですむ。この命令は、8ビット・レジスタあるいはメモリをインクリメントするために用いられる。
4. この命令は、キャリー・フラグに影響を与えない。

INC reg (Increment)

レジスタをインクリメントする。

指定されたレジスタの内容に1を加える。これは16ビットのインクリメント命令である。

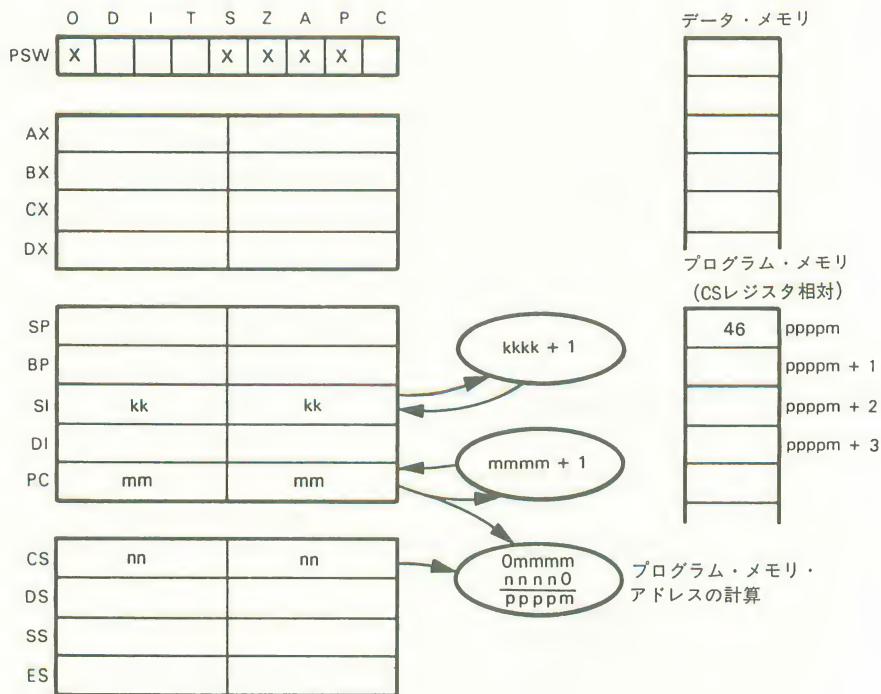


`rrr=000` : A X
`001` : C X
`010` : D X
`011` : B X
`100` : S P
`101` : B P
`110` : S I
`111` : D I

S I レジスタの内容が $00FF_{16}$ の場合を考える。

INC SI

の実行によって、S I レジスタの内容は 0100_{16} に増加する。



INC SI

サイクル数:2

注)

1. この命令は、8080の命令 `INC reg` と同じ機能を果たす。
2. セグメント・レジスタは、この命令を用いてインクリメントすることはできない。
3. この命令は、キャリー・フラグに影響を与えない。

INT (Interrupt)

ソフトウェアでインタラプトを発生させる。

この命令は、次の一連の動作を行なう。

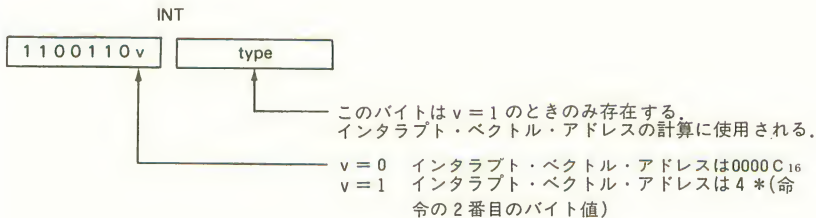
1. フラグ・レジスタをスタックにプッシュする。
2. IFとTFのフラグを0にクリアする。
3. CSレジスタをスタックにプッシュする。
4. メモリ・アドレス $00xxx_{16}$ のワードをCSレジスタにロードする。xxx は、オペ・コードの最下位ビットと必要ならば命令の2番目のバイトによって決定される。オペ・コードの最下位ビットが0ならば、xxx は $00E_{16}$ となる。オペ・コードの最下位ビットが1ならば、xxx は命令の2番目のバイトの値を4倍して2を加えたものに等しい。言い換えれば、次のように書ける。

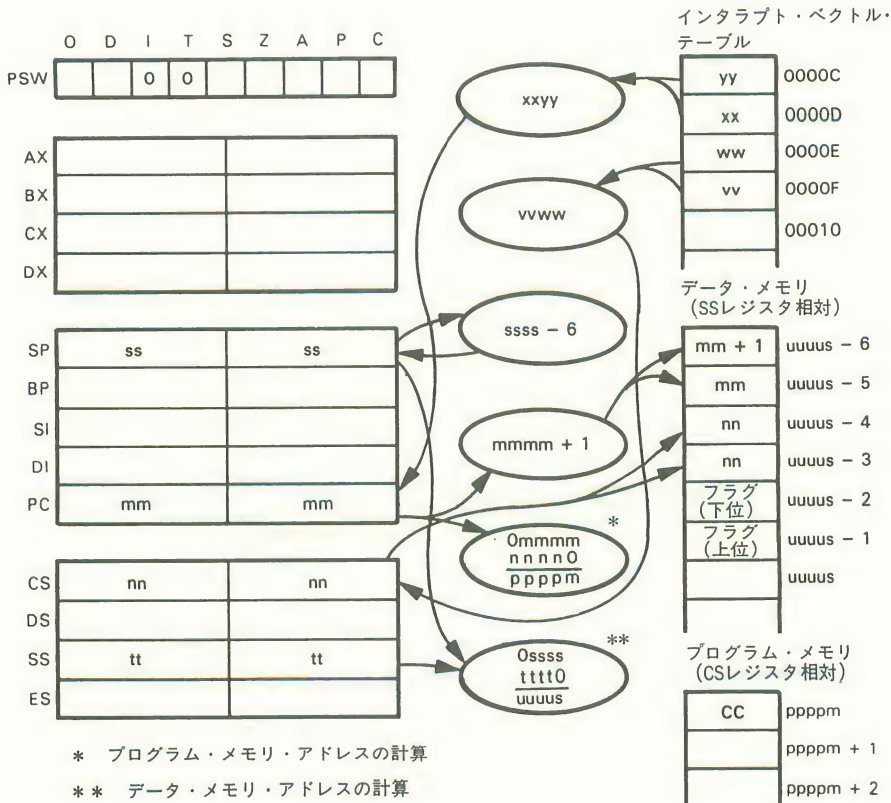
```
IF 最下位ビット = 0 THEN xxx =  $00E_{16}$ 
ELSE xxx =  $(4 * 2 \text{ 番目のバイト}) + 2$ 
```

5. PCレジスタをスタックにプッシュする。
6. メモリ・アドレス $00yyy_{16}$ のワードをPCレジスタにロードする。yyy は、命令コードの最下位ビットと必要ならば命令の2番目のバイトによって決定される。命令コードの最下位ビットが0ならば、yyy は $00C_{16}$ となる。命令コードの最下位ビットが1ならば、yyy は命令の2番目のバイトの値を4倍したものに等しい。言い換えれば、次のように書ける。

```
IF 最下位ビット = 0 THEN yyy =  $00C_{16}$ 
ELSE yyy =  $4 * 2 \text{ 番目のバイト}$ 
```

命令コードを次に示す。





INT
サイクル数: 52: v = 0 のとき
51: v = 1 のとき

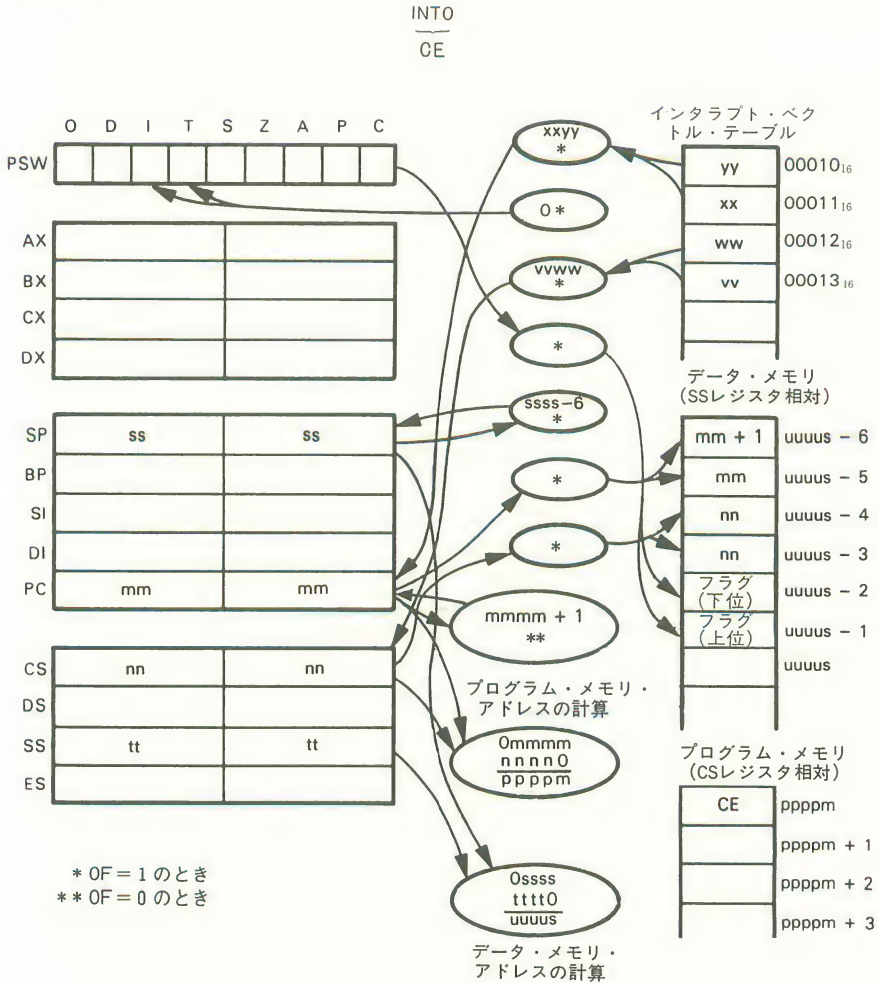
INTO (Interrupt if Overflow)

OF = 1 ならば、タイプ4のインタラプトを発生させる。

OF = 0 ならば、この命令は無操作となる。OF = 1 ならば、次の一連の動作を行なう。

1. フラグ・レジスタをスタックにプッシュする。
2. IFとTFのフラグを0に設定する。
3. CSレジスタをスタックにプッシュする。
4. メモリ位置00012₁₆のワードをCSレジスタに移動する。
5. PCレジスタをスタックにプッシュする。
6. メモリ位置00010₁₆のワードをPCレジスタに移動する。

この位置から実行を再開する。
命令コードを次に示す。



INTO

サイクル数: 53: OF = 1

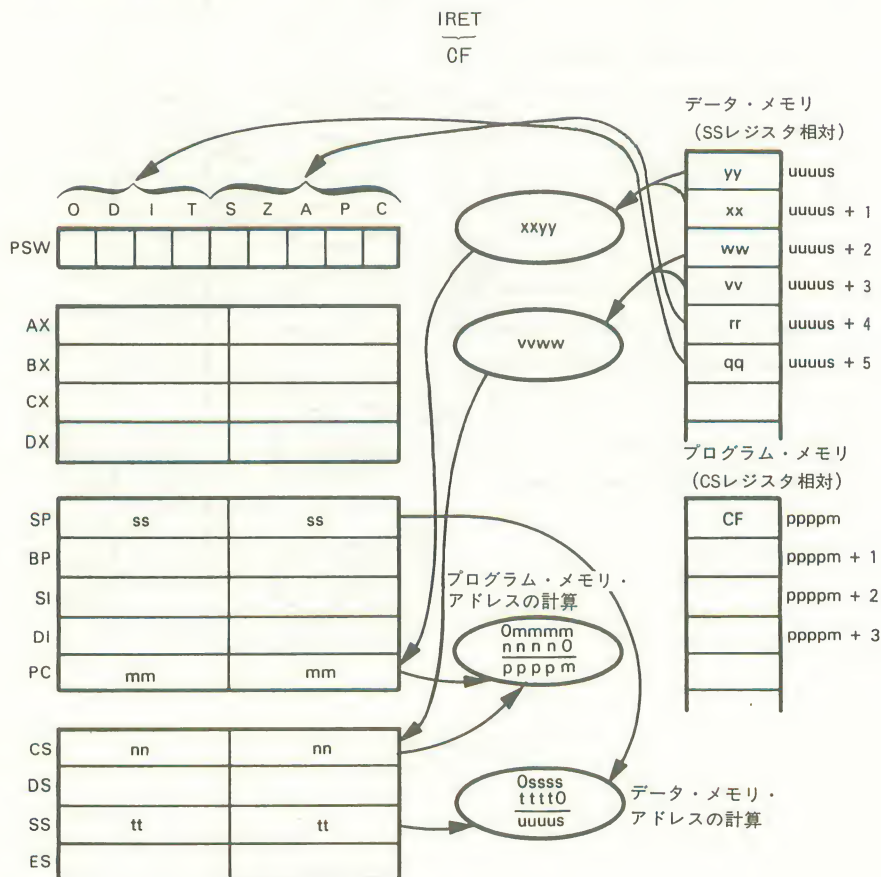
4: OF = 0

IRET (Interrupt Return)

インタラプト処理からリターンする。

スタックのトップの2バイトをプログラム・カウンタにポップする。この2バイトは、次に実行される命令のオフセット・アドレスを与える。スタックの次の2バイトをCSレジスタにポップする。この2バイトは、次に実行される命令のコード・セグメント・アドレスを与える。スタックの次の2バイトをフラグ・レジスタにポップする。以前のプログラム・カウンタ、コード・セグメントとフラグのレジスタの内容は失なわれる。

命令コードを次に示す。



IRET

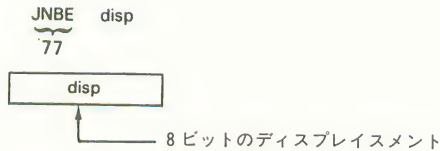
サイクル数: 32

JA disp (Jump if Above)
 JNBE disp (Jump if Not Below nor Equal)

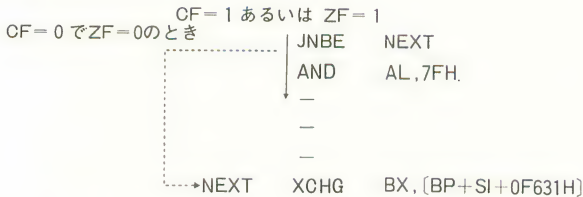
上ならばジャンプする。／下でも等しくもなければジャンプする。

この命令は、キャリー・フラグとゼロ・フラグが共に0の場合だけジャンプが行なわれ、それ以外では次の命令が実行されることを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JNBE 命令に続いて、キャリー・フラグとゼロ・フラグが0 ならばXCHG 命令が実行される。キャリー・フラグあるいはゼロ・フラグが1 ならば、AND 命令が実行される。

サイクル数：分岐したとき：16

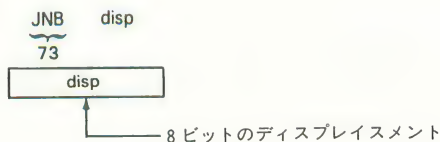
分岐しなかったとき：4

JAE disp (Jump if Above or Equal)
 JNB disp (Jump if Not Below)

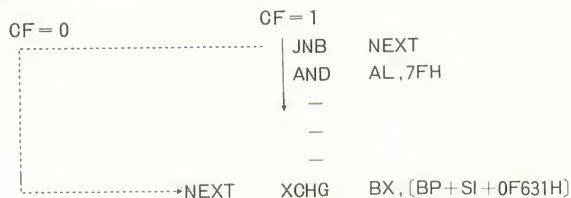
上あるいは等しければジャンプする。／下でなければジャンプする。

この命令は、キャリー・フラグが0 の場合だけジャンプが行なわれ、それ以外では次の命令が実行されることを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JNB 命令に続いて、キャリー・フラグが0 ならばXCHG 命令が実行される。キャリー・フラグが1 ならば、AND 命令が実行される。

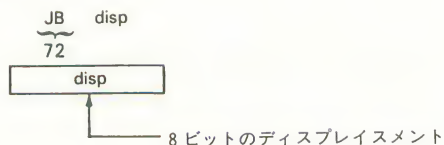
サイクル数：分岐したとき：16
分岐しなかったとき：4

JB disp (Jump if Below)
JNAE disp (Jump if Not Above nor Equal)

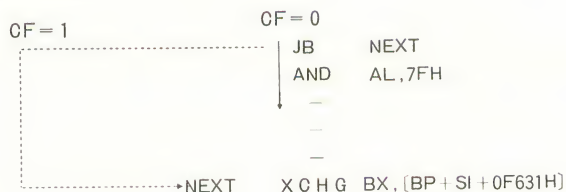
下ならばジャンプする。／上でも等しくもなければジャンプする。

この命令は、キャリー・フラグが1 の場合だけジャンプが実行されることを除けば、JMP 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



J B 命令に続いて、キャリー・フラグが 1 ならば X C H G 命令が実行される。キャリー・フラグが 0 の場合は A N D 命令が実行される。

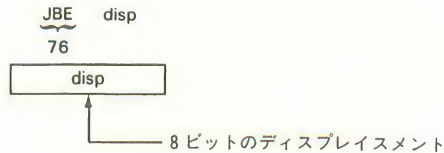
サイクル数：分岐したとき：16
分岐しなかったとき：4

JBE disp (Jump if Below or Equal)
JNA disp (Jump if Not Above)

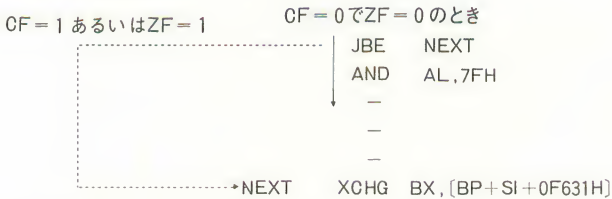
下あるいは等しければジャンプする。／上でなければジャンプする。

この命令は、キャリー・フラグあるいはゼロ・フラグが 1 ならばジャンプし、そうでなければ次の命令を実行することを除いて、J M P disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



J B E 命令に続いて、キャリー・フラグあるいはゼロ・フラグが 1 ならば X C H G 命令が実行される。キャリー・フラグとゼロ・フラグが共に 0 ならば、A N D 命令が実行される。

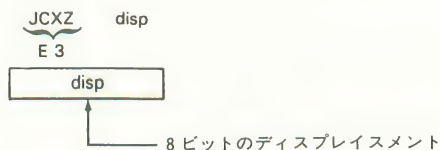
サイクル数：分岐したとき：16
分岐しなかったとき：4

JCXZ disp (Jump if CX is Zero)

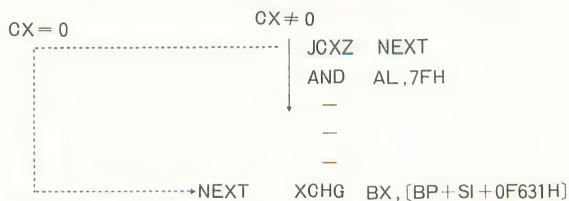
C X が 0 ならばジャンプする。

この命令は、C X レジスタが 0 の場合だけジャンプし、それ以外では次の命令を実行することを除いて、J M P disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JCXZ 命令に続いて、CX レジスタが 0 ならば XCHG 命令が実行される。CX レジスタが 0 でなければ、AND 命令が実行される。

この命令は CX の 0 を判断するためにゼロ・フラグを参照するのではなく、CX レジスタを直接参照することに注意。

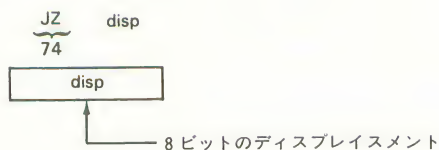
サイクル数：分岐した時：18
分岐しなかったとき：6

JE disp (Jump if Equal)
JZ disp (Jump if Zero)

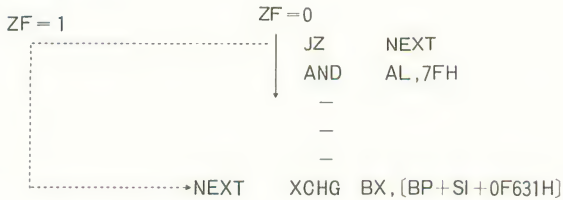
等しければジャンプする。/ 0 ならばジャンプする。

この命令は、ゼロ・フラグが 1 ならばジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JZ 命令に続いて、ゼロ・フラグが1 ならばXCHG 命令が実行される。ゼロ・フラグが0 ならば、AND 命令が実行される。

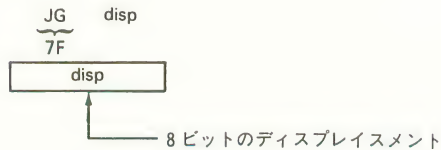
サイクル数：分岐したとき：16
分岐しなかったとき：4

JG disp (Jump if Greater)
JNLE disp (Jump if Not Less nor Equal)

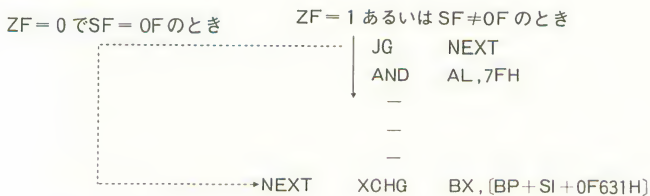
大きければジャンプする。／小さくもなく等しくもなければジャンプする。

この命令は、ゼロ・フラグが0 でしかもサイン・フラグとオーバーフロー・フラグが等しければジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JG 命令に続いて、ゼロ・フラグが0 でサイン・フラグとオーバーフロー・フラグが等しければ、XCHG 命令が実行される。ゼロ・フラグが1 あるいはサイン・フラグとオーバーフロー・フラグが等しくなければ、AND 命令が実行される。

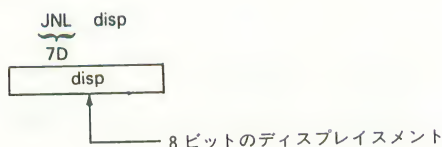
サイクル数：分岐したとき：16
分岐しなかったとき：4

JGE disp (Jump if Greater or Equal)
JNL disp (Jump if Not Less)

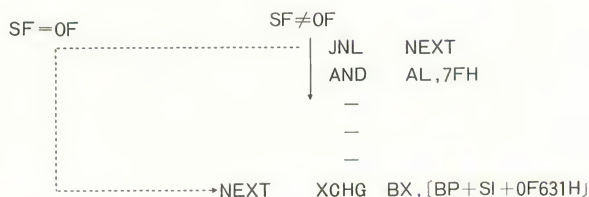
大きいか等しければジャンプする./小さくなければジャンプする.

この命令は、サイン・フラグがオーバーフロー・フラグに等しければジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JNL 命令に続いて、サイン・フラグがオーバーフロー・フラグに等しければXCHG 命令が実行される。サイン・フラグがオーバーフロー・フラグに等しくなければ、AND 命令が実行される。

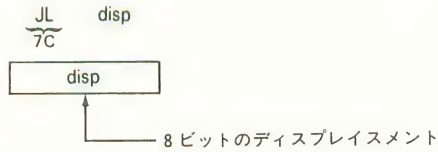
サイクル数：分岐したとき：16
分岐しなかったとき：4

JL disp (Jump if Less)
JNGE disp (Jump if Not Greater nor Equal)

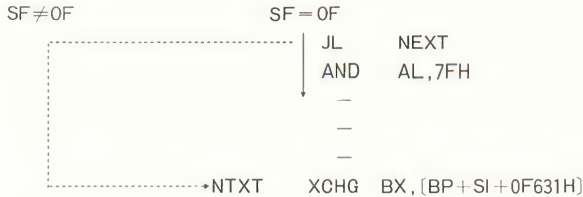
小さければジャンプする./大きくもなく等しくもなければジャンプする。

この命令は、サイン・フラグがオーバーフロー・フラグに等しくなければジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JL 命令実行後、サイン・フラグがオーバーフロー・フラグに等しくなければ XCHG 命令が実行される。サイン・フラグとオーバーフロー・フラグが等しければ、AND 命令が実行される。

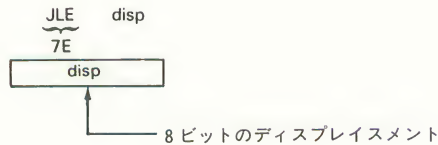
サイクル数：分岐したとき：16
分岐しなかったとき：4

JLE disp (Jump if Less or Equal)
JNG disp (Jump if Not Greater)

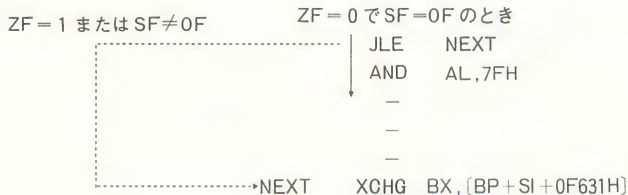
小さいか等しければジャンプする。／大きくなければジャンプする。

この命令は、ゼロ・フラグが1 かあるいはサイン・フラグがオーバーフロー・フラグに等しくなければジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JLE 命令に続いて、ゼロ・フラグが1あるいはサイン・フラグがオーバーフロー・フラグに等しくなければ、XCHG 命令が実行される。ゼロ・フラグが0でサイン・フラグがオーバーフロー・フラグに等しければ、AND 命令が実行される。

サイクル数：分岐したとき：16

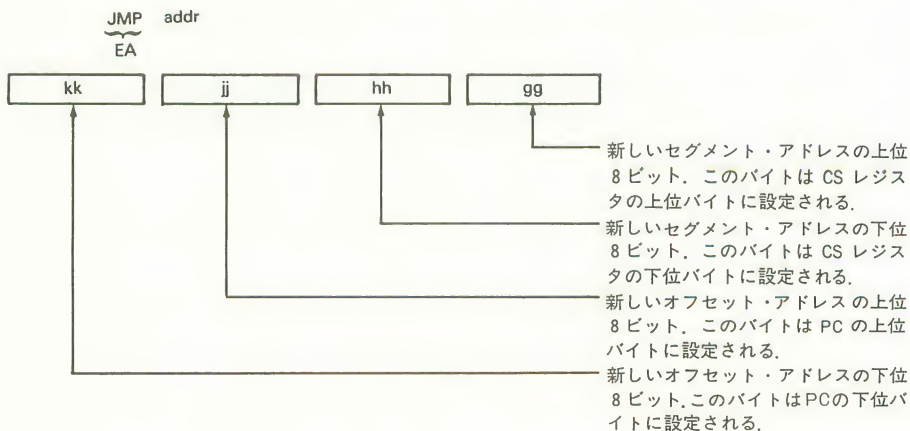
分岐しなかったとき：4

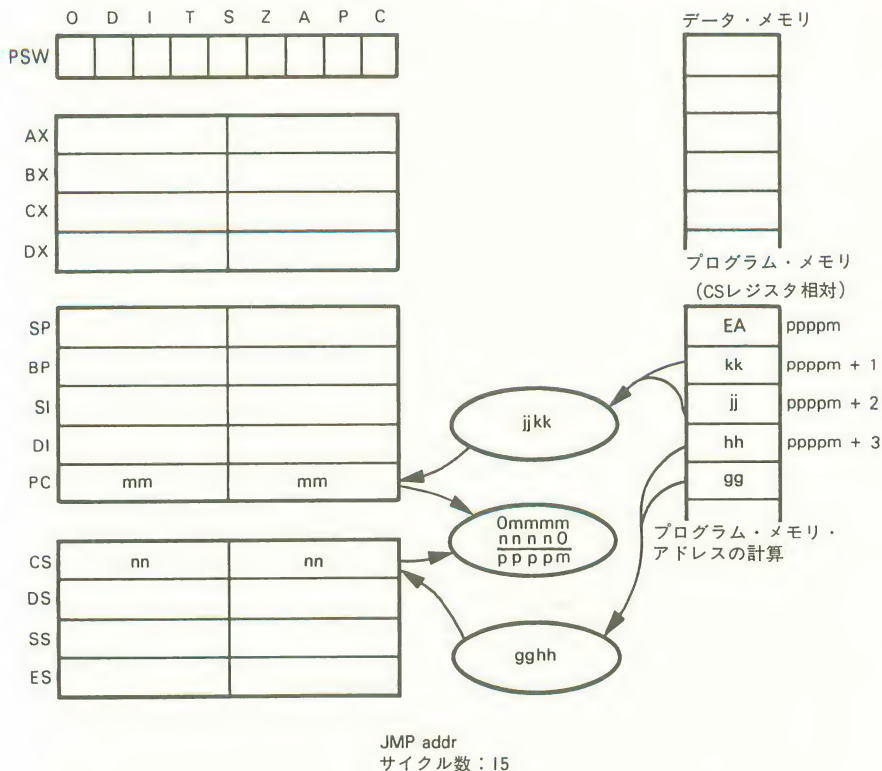
JMP addr (Jump)

オペランドで指定された位置にジャンプする。

後続のプログラム・メモリの2バイトの内容をPCレジスタへ移動する。その次のプログラム・メモリの2バイト（命令のバイト4と5）の内容をCSレジスタに移動する。この位置から実行を続ける。以前のプログラム・カウンタとコード・セグメント・レジスタの内容は失われる。

命令コードを次に示す。





注)

1. イミディエイトの32ビットは目標のアドレスで、遠い位置のラベル（セグメント・レジスタを変える）と考えられる。
2. 遠い位置のラベルへ直接の移動が行なわれる。

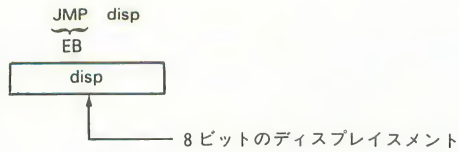
JMP disp (Jump)

オペランドで指定された位置にジャンプする。

この命令は、2番目のオブジェクト・コード・バイト（符号付きの8ビット・ディスプレイメントと見なす）の内容を、プログラム・カウンタに2を加えた内容に加算する*。これは、次に実行される命令のオフセット・アドレスになる。以前のプログラム・カウンタの内容は失なわれる。コード・セグメント・レジスタの内容は変化しない。

命令コードを次に示す。

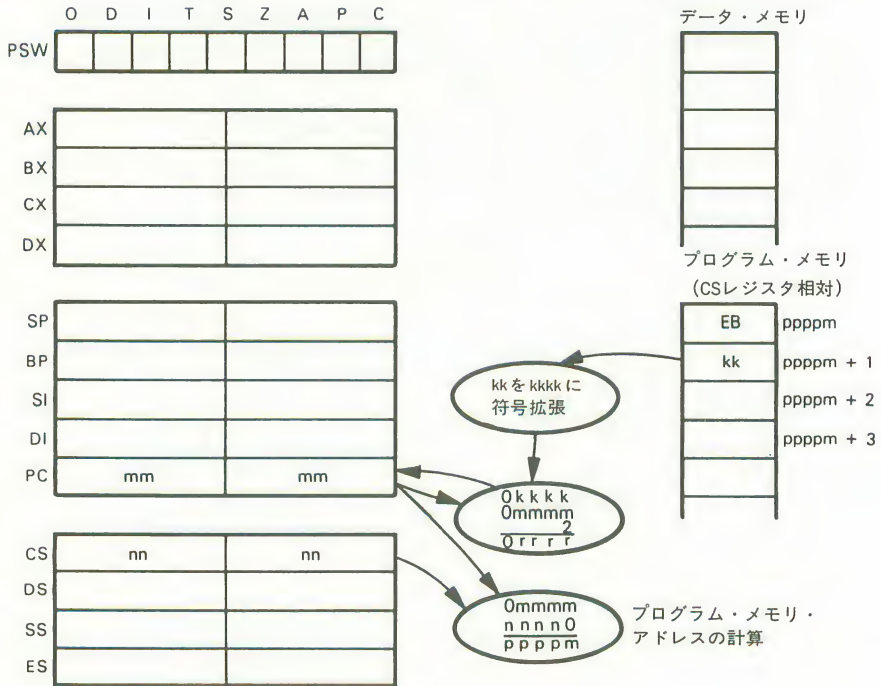
* 2を加えるのは、この命令が2バイトで構成されているため（訳者注）。



次の一連の命令において、

	JMP	NEXT
	AND	AL, 7FH
	—	
	—	
	—	
NEXT	XOR	AL, 7FH

JMP 命令に続いて、XOR 命令が実行される。AND 命令は、どこかでジャンプあるいはコールの命令でこの命令に分岐しない限りは決して実行されない。



JMP kk
サイクル数：15

注)

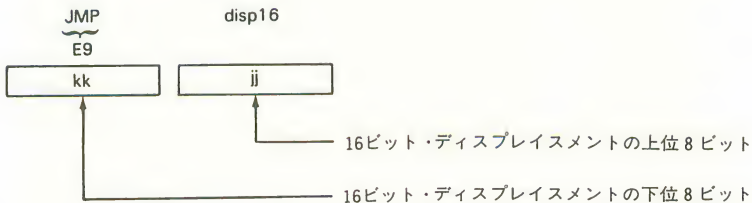
1. この命令は、プログラム相対アドレス指定を用いている。これは、An Introduction to Microcomputers: Volume I—Basic Concepts (Osborne/McGraw-Hill, 1978) に述べられているプログラム相対ページングと類似している。異なっているのは、8ビットの符号付きディスプレイスメントが加算される前に、プログラム・カウンタの内容は次の命令を指すために増加することである。
2. これは、現在のセグメント内の自己相対の分岐である。
3. 8ビットのディスプレイスメントは近い位置のラベルのディスプレイスメントと考えられる。

JMP disp16 (Jump)

オペランドで指定された位置にジャンプする。

後続の2つのプログラム・メモリ・バイトの内容を、16ビットの符号なしディスプレイスメントと見なして、プログラム・カウンタに加算する。この位置から実行を続ける。以前のプログラム・カウンタの内容は失われる。

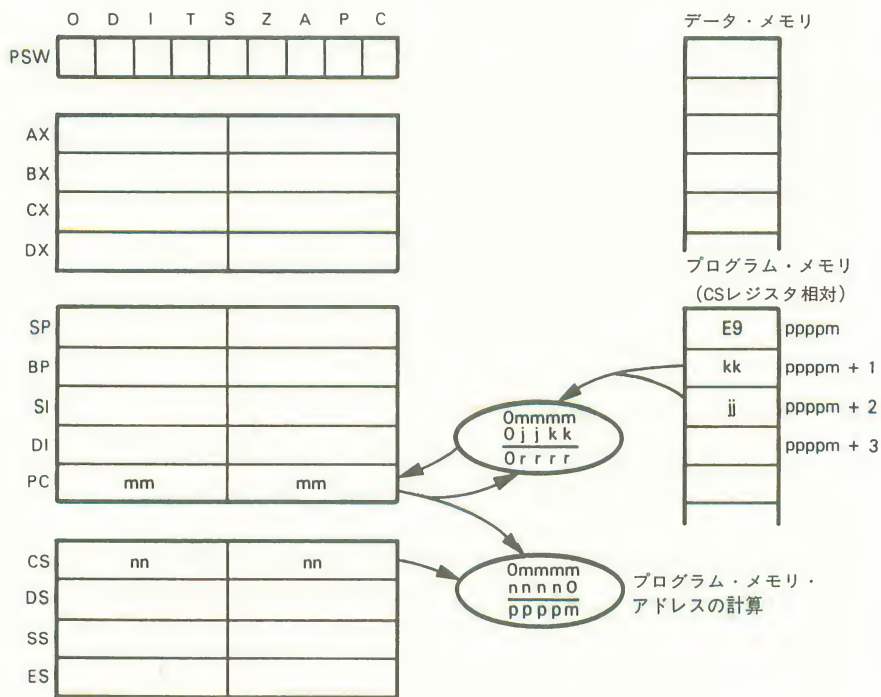
命令コードを示す。



次の一連の命令において、

	JMP	NEXT
BRICKS	AND	AL, 7FH
	—	
	—	
	—	
NEXT	STOS	BYTE

JMP 命令実行後、STOS 命令が実行される。AND 命令は、処理のどこかでCALLあるいはJMP 命令で BRICKS がそのオペランドとして参照されない限りは、実行されることはない。



JMP jkk
サイクル数: 15

注)

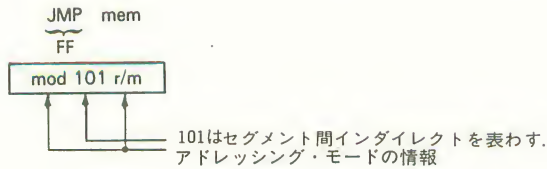
1. これは、現在のセグメント内での自己相対の分岐である。
2. 16ビットのディスプレイスメントは、近い位置のラベル（このセグメント内）のディスプレイスメントと考えられる。

JMP mem (Jump)

オペランドで指定された位置にジャンプする。

指定されたメモリ位置のワードをプログラム・カウンタへ移動し、続くワードをCSレジスタに移動する。この位置から実行を続ける。以前のプログラム・カウンタとコード・セグメント・レジスタの内容は失われる。

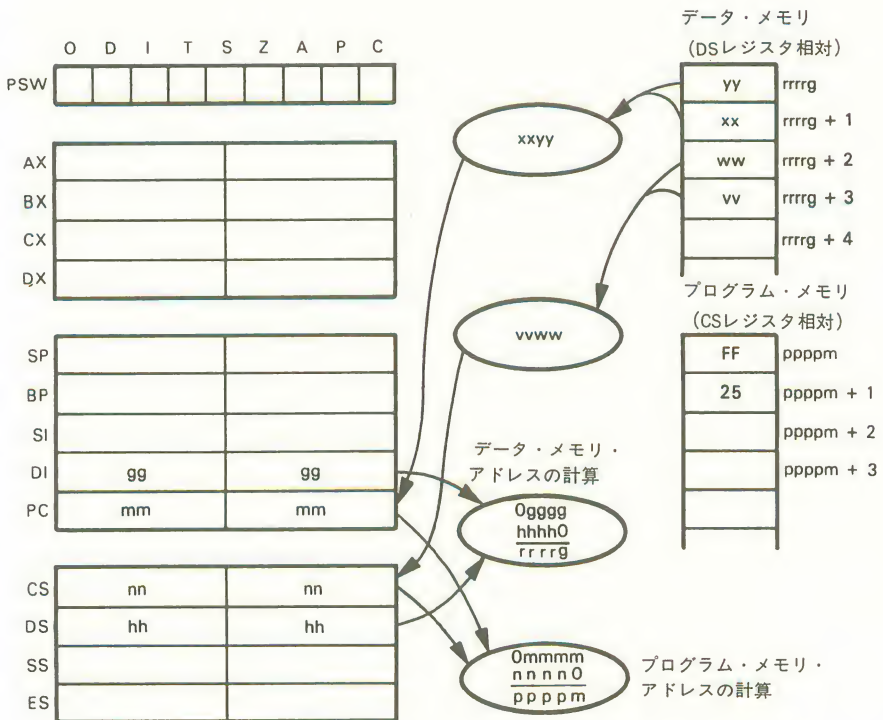
命令コードを次に示す。



D S レジスタが 7000_{16} を含み、D I レジスタが 0404_{16} を含み、メモリ位置 70404_{16} のワードが 1000_{16} でメモリ位置 70406_{16} のワードが $7E00_{16}$ であると仮定する。

JMP far-ptr [DI]

の実行後、プログラム・カウンタは 1000_{16} に、C S レジスタは $7E00_{16}$ になる。命令の実行は $7F000_{16}$ から続けられる。



JMP far_ptr[DI]

サイクル数：24+EA セグメント間

注)

- この命令では、レジスタ・アドレッシングは有効でない。

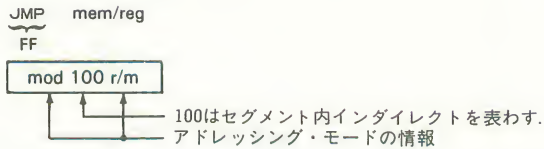
2. これは、メモリを用いたセグメント間（セグメントを変える）インダイレクト・ジャンプであり、ジャンプ・テーブルによく用いられる。
3. 32ビットの目的アドレスは、遠い位置のラベルと考えられる。
4. [D I]の前に `far-ptr` ディレクティブを書くことによって、アセンブラはメモリ中の32ビット・ポインタを用いる `JMP` 命令を生成する。

JMP mem/reg (Jump)

オペランドで指定された位置にジャンプする。

指定されたオペランドがレジスタならば、そのレジスタの内容をプログラム・カウンタに移動する。指定されたオペランドがメモリ位置ならば、そのメモリ位置の内容をプログラム・カウンタに移動する。この位置から実行を続ける。以前のプログラム・カウンタの内容は失われる。CSレジスタは変化しない。

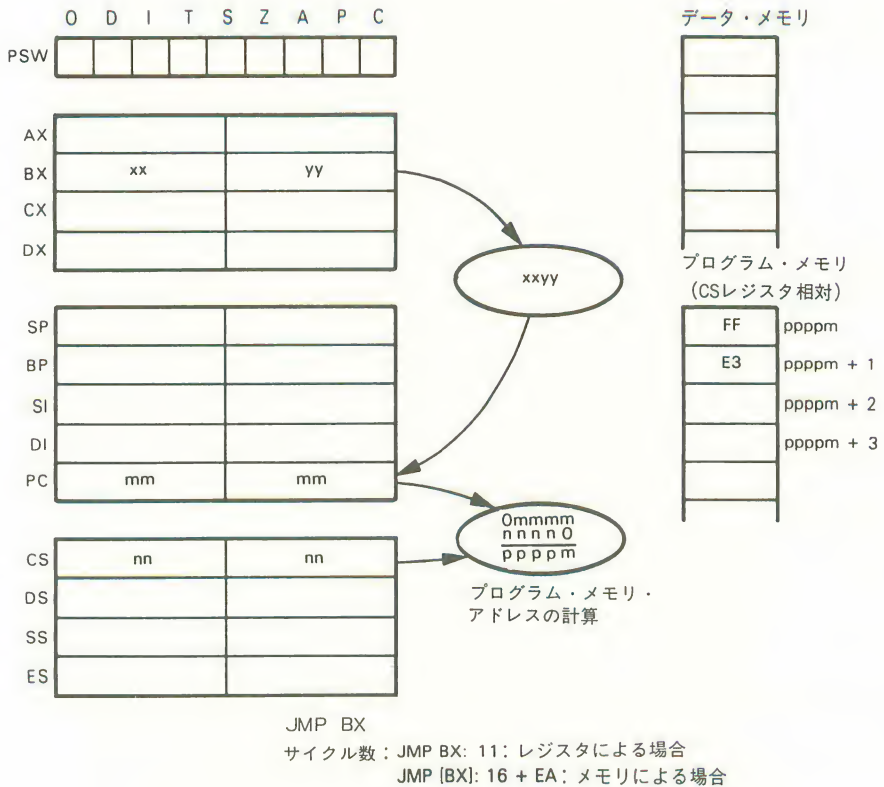
命令コードを次に示す。



BXレジスタが $14A9_{16}$ を含むと仮定する。

`JMP BX`

の実行後、PCは $14A9_{16}$ となり、 $14A9_{16}$ を次の命令のオフセット・アドレスとして、実行を再開する。



注)

1. これは、メモリあるいはレジスタによるセグメント内インダイレクト・ジャンプである。
2. レジスタあるいはステータスは影響を受けない。
3. 16ビットの目的アドレスは、近い位置のラベル（セグメント内）と考えられる。
4. far-ptr がないことで、前の JMP 命令のときの32ビット・ポインタではなくて、メモリの16ビット・ポインタを表わしている。このアセンブラの変換で、1つのニーモニック JMP をさまざまな2進命令コードに用いることができる。

JNE disp (Jump if Not Equal)
JNZ disp (Jump if Not Zero)

等しくなければジャンプする。／0でなければジャンプする。

この命令は、ゼロ・フラグが0ならばジャンプし、それ以外では次の命令を実行すること

JNO命令に続いて、オーバーフロー・ステータスが0ならば、XCHG命令が実行される。オーバーフロー・ステータスが1ならば、AND命令が実行される。

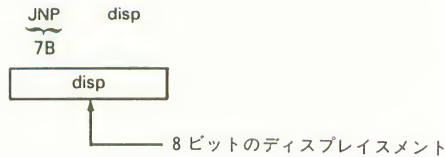
サイクル数：分岐したとき：16
分岐しなかったとき：4

JNP disp (Jump if Not Parity)
JPO disp (Jump if Parity Odd)

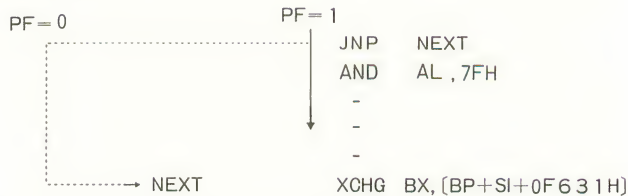
パリティ・フラグが0ならばジャンプする。／パリティが奇数ならばジャンプする。

この命令は、パリティ・フラグが0ならばジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



JNP命令に続いて、パリティ・フラグが0ならばXCHG命令が実行される。パリティ・フラグが1ならば、AND命令が実行される。

サイクル数：分岐したとき：16
分岐しなかったとき：4

JNS disp (Jump if Not Sign)

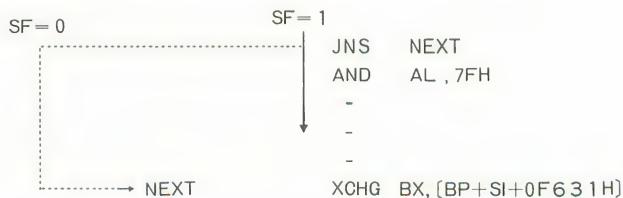
サイン・フラグが0ならばジャンプする。

この命令は、サイン・フラグが0ならばジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す.



次の一連の命令において,



JNS 命令実行後、サイン・フラグが 0 ならば XCHG 命令が実行され、そうでなければ AND 命令が実行される。

サイクル数：分岐したとき：16

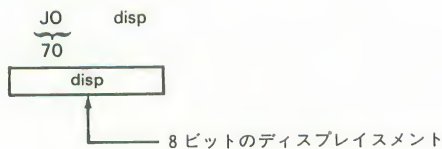
分岐しなかったとき：4

JO disp (Jump if Overflow)

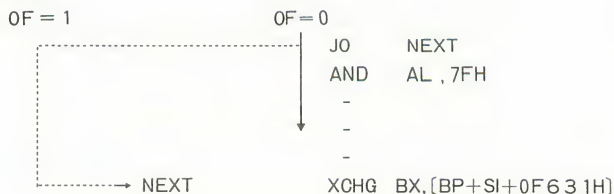
オーバーフローならばジャンプする。

この命令は、オーバーフロー・フラグが 1 ならばジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において,



J O 命令に続いて、オーバーフロー・フラグが1ならばXCHG命令が実行される。オーバーフロー・フラグが0ならば、AND命令が実行される。

サイクル数：分岐したとき：16

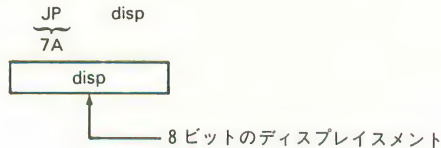
分岐しなかったとき：4

JP disp (Jump if Parity)
JPE disp (Jump if Parity Even)

パリティ・フラグが1ならばジャンプする。／パリティが偶数ならばジャンプする。

この命令は、パリティ・フラグが1ならばジャンプし、それ以外では次の命令を実行することを除いて、JMP disp 命令と同じである。

命令コードを次に示す。



次の一連の命令において、



J P 命令に続いて、パリティ・フラグが1ならばXCHG命令が実行される。パリティ・フラグが0ならば、AND命令が実行される。

サイクル数：分岐したとき：16

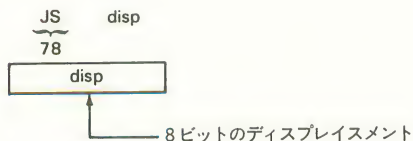
分岐しなかったとき：4

JS disp (Jump if Sign)

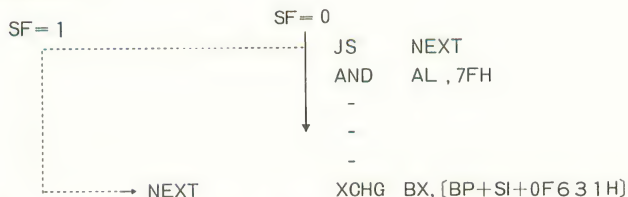
サイン・フラグが1ならばジャンプする。

この命令は、サイン・フラグが1ならばジャンプすることを除いて、JMP disp 命令と同じである。

命令コードを次に示す.



次の一連の命令において,



J S 命令に続いて, サイン・フラグが 1 ならば X C H G 命令が実行される. サイン・フラグが 0 ならば, A N D 命令が実行される.

サイクル数: 分岐したとき: 16

分岐しなかったとき: 4

LAHF (Load AH from 8080 Flags)

8080のフラグをA Hレジスタにロードする.

この命令は, フラグ・レジスタの下位 8 ビットをA Hレジスタに移動する. 移動される 8 ビットを次に示す.

7	6	5	4	3	2	1	0
SF	ZF	X	AF	X	PF	X	CF

ここで, Xは不確定の値を表わす.

命令コードを次に示す.

LAHF
9F

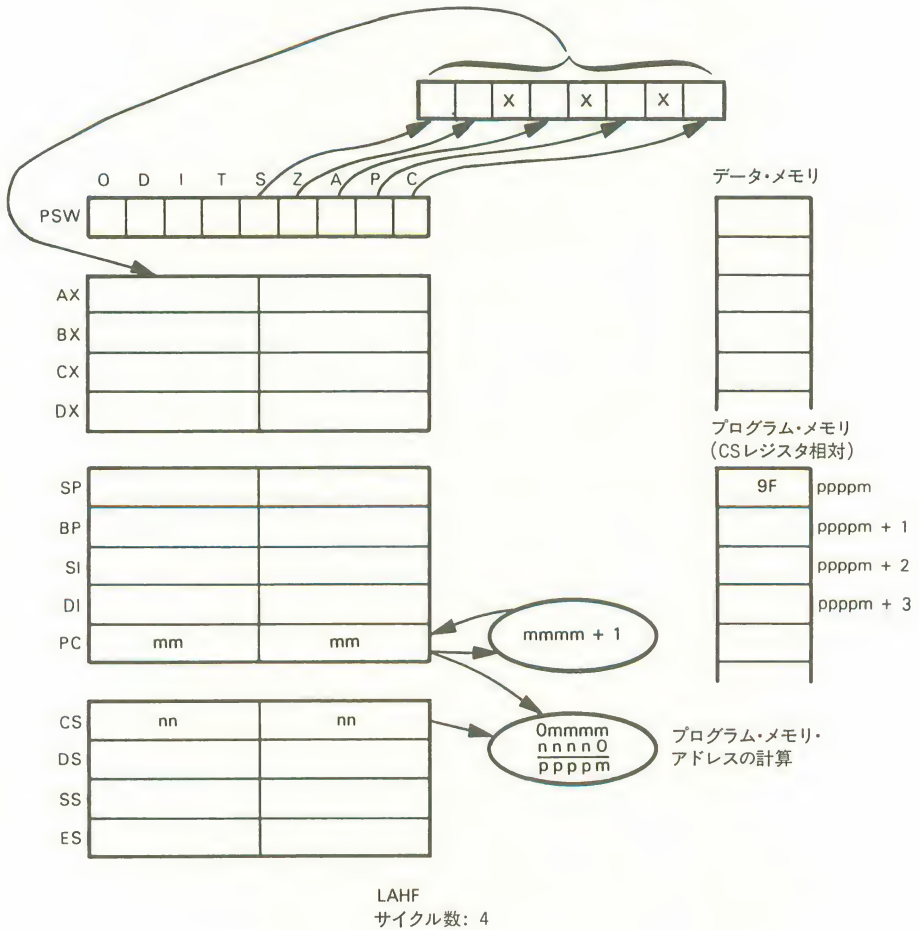
例として, C FとP Fのフラグが1で, Z F, S F, A Fのフラグが0の場合を考える.

L A H F

を実行すると, A Hレジスタの内容は,

0 0 X 0 X 1 X 1

になる.



注)

1. ステータスは影響を受けない. レジスタはAHを除いて影響を受けない.
2. この命令は, 8080の命令 PUSH PSW をエミュレートするために, PUSH AX と共に用いられる.

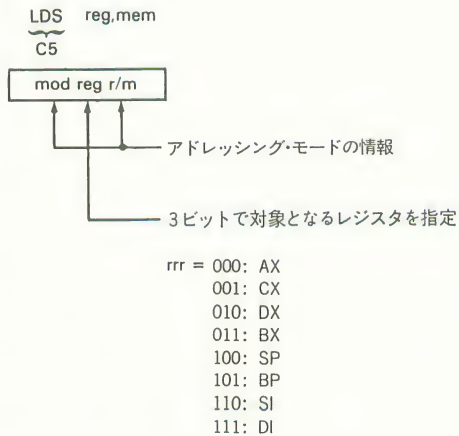
8086コード	8080コード
LAHF	PUSH PSW
PUSH AX	

LDS reg,mem (Load register and DS)

メモリからレジスタとDSにロードする。

指定されたメモリ・ワードを指定されたレジスタにロードし、それに続くメモリ・ワードをDSレジスタにロードする。

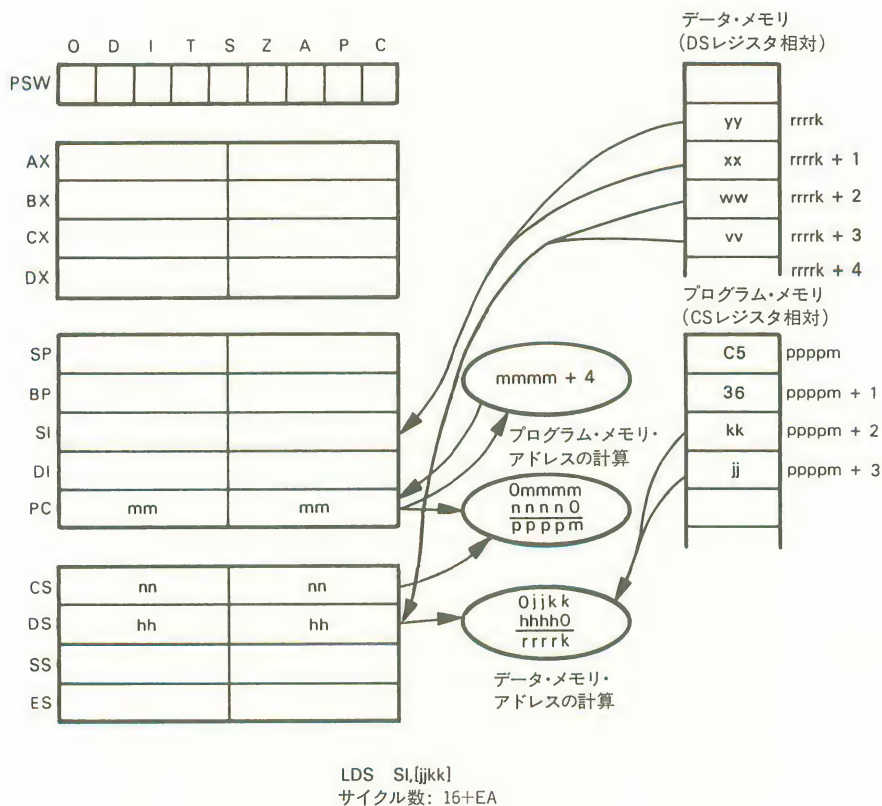
命令コードを次に示す。



例として、DSレジスタがC000₁₆を含み、メモリ位置C0010₁₆のワードが0180₁₆で、メモリ位置C0012₁₆のワードが2000₁₆の場合を考える。

LDS SI,[0H]

の実行後、SIレジスタは0180₁₆に、DSレジスタは2000₁₆になる。



注)

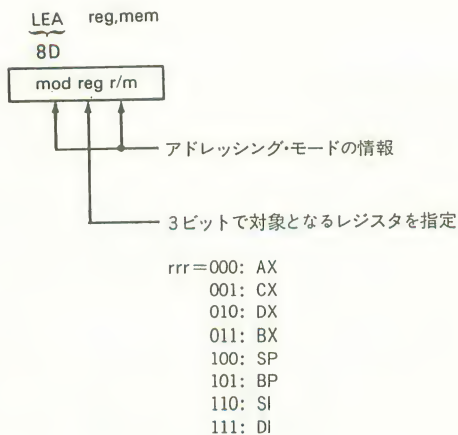
1. ステータスは影響を受けない。
2. mod が11の場合、この命令による動作は定義されない。

LEA reg,mem (Load Effective Address)

オフセット・アドレスをレジスタにロードする。

メモリ・オペランドを指定する16ビットのオフセット・アドレスを、指定されたレジスタにロードする。

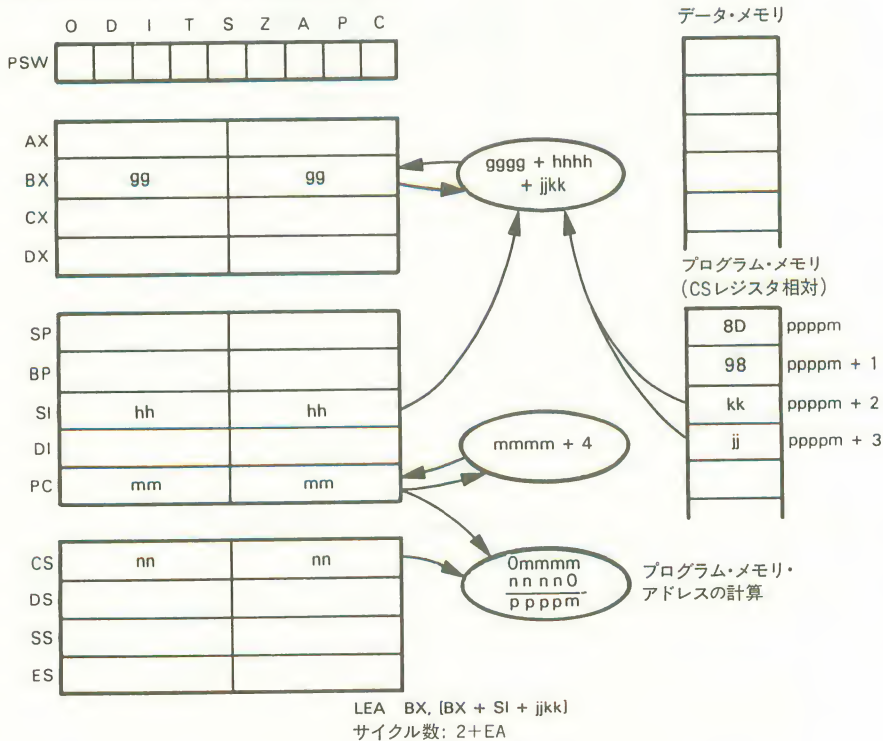
命令コードを次に示す。



BXレジスタが 0400_{16} を含み、SIレジスタが $003C_{16}$ を含むと仮定する。

LEA BX, [BX + SI + 0F62H]

の実行後、BXレジスタは $139E_{16}$ になる。この値は、BXとSIレジスタの内容と指定されたディスプレイメントの和である。



注)

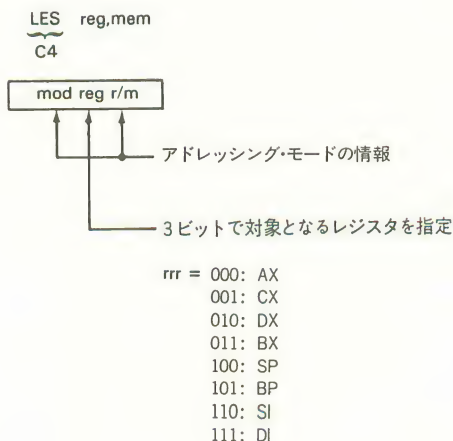
1. ステータスは影響を受けない.
2. mod が11の場合, この命令による動作は定義されない.

LES reg, mem (Load register and ES)

メモリからレジスタとESにロードする.

指定されたメモリ・ワードを指定されたレジスタにロードし, それに続くメモリ・ワードをESレジスタにロードする.

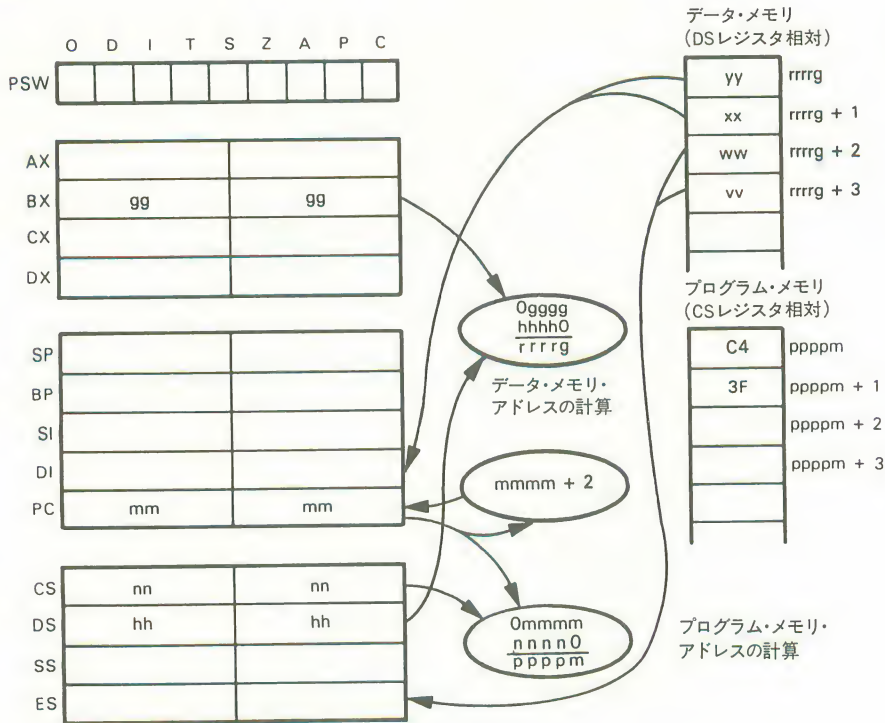
命令コードを次に示す.



DSレジスタが $B000_{16}$ を含み, BXレジスタが $080A_{16}$ を含み, $B080A_{16}$ におけるメモリ・ワードが $05A2_{16}$ で, $B080C_{16}$ のメモリ・ワードが 4000_{16} であると仮定する.

LES DI, [BX]

の実行後, DIレジスタは $05A2_{16}$ に, ESレジスタは 4000_{16} になる.



LES DI,[BX]
サイクル数: 16+EA

注)

1. ステータスは影響を受けない。
2. mod が11の場合、この命令による動作は定義されない。
3. DIレジスタは本来ESレジスタと関連したレジスタなので、この命令で指定されるレジスタとしてはDIレジスタが一般的である。

LOCK (Lock)

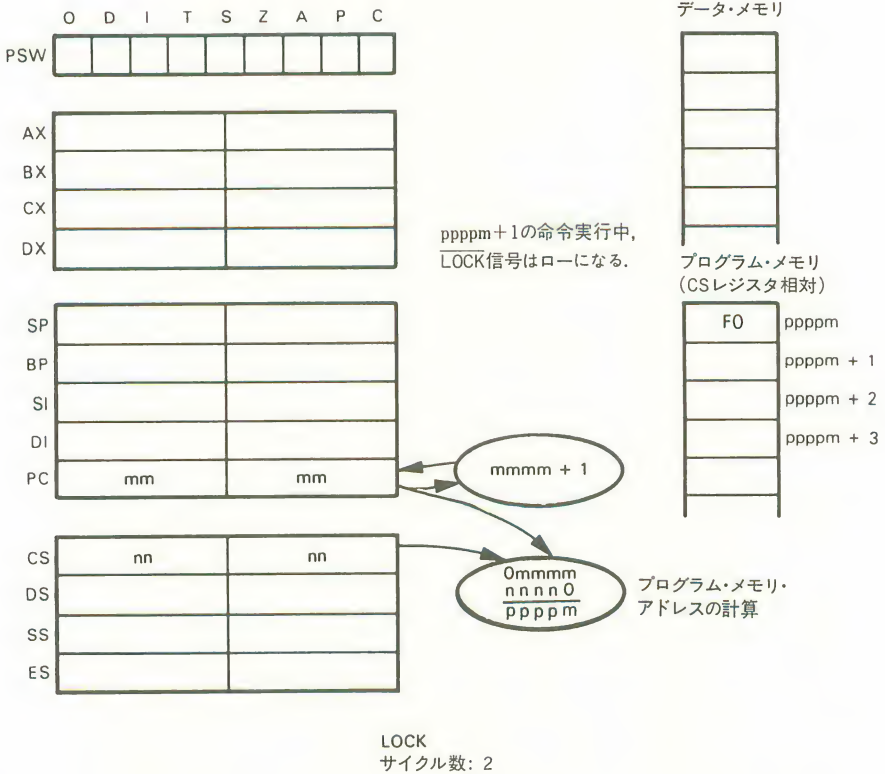
バスのロック信号を設定する。

この命令は、8086にローのLOCK信号を出力させるために用いられる。LOCK信号は、次の命令実行中はローに保たれる。

この命令は、プレフィックス命令と考えられる。すなわち、LOCK信号の設定を必要とする命令に先行して用いられる。

命令コードを次に示す.

LOCK
F0



注)

1. このプレフィックスは、任意の8086命令に前置して用いられる。しかし、このプレフィックスがREPプレフィックスとストリング・プリミティブと共に用いられた場合は問題となる。この詳細については次章を参照のこと。
2. このプレフィックスは、テスト・アンド・セットのシーケンス実行に有用である。

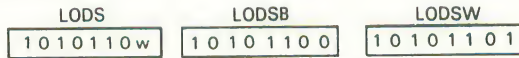
LODS/LODSB/LODSW (Load String)

メモリからALあるいはAXのレジスタにロードする。

SIレジスタで示されるメモリの内容を、AL (8ビット操作) あるいはAX (16ビット

ト操作) のレジスタに移動する。S I レジスタは、D F フラグの値によって増加あるいは減少する。

命令コードを次に示す。



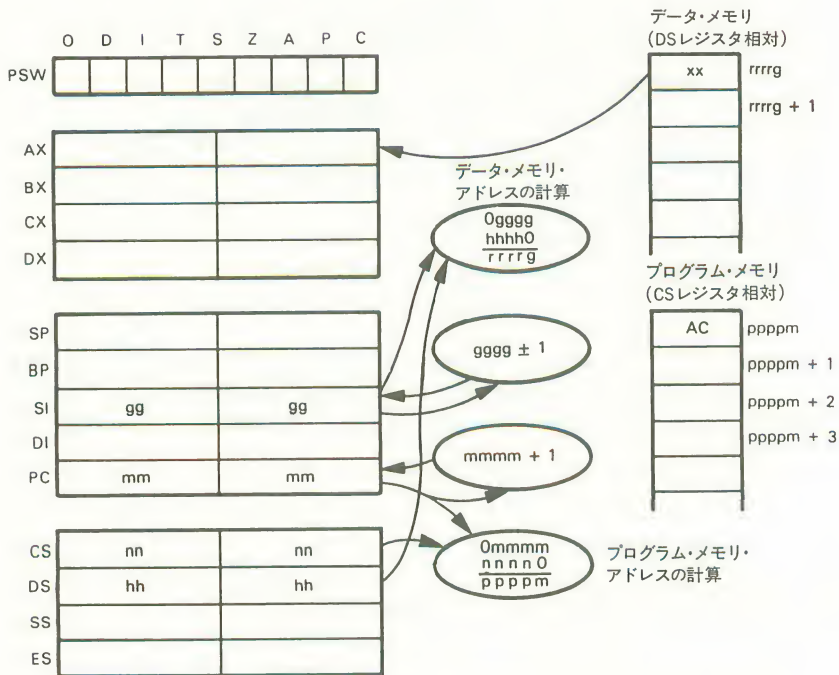
w=0 8ビットの転送
DF=0ならばSIレジスタの値は
1だけ増加、そうでなければ1
だけ減少する。

w=1 16ビットの転送
DF=0ならばSIレジスタの値は
2だけ増加、そうでなければ2
だけ減少する。

たとえば、D F フラグが0で、S I レジスタが0035₁₆を含み、D S レジスタが4008₁₆を含み、メモリ位置400B5₁₆のバイトが0F₁₆であると仮定する。

LODSB

の実行後、A L レジスタの内容は0F₁₆になり、S I レジスタの内容は0036₁₆になる。



LODSB

サイクル数: 12: 1度だけ実行のとき

$9 + (13 * R)$: REP プレフィックスによってR回実行されたとき

注)

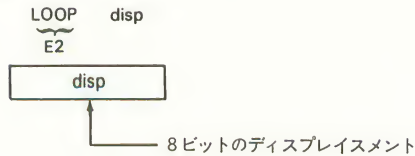
1. ステータスは影響を受けない。
2. デフォルト・セグメント・レジスタはDSレジスタである。これは、適当なセグメント変更プレフィックスによって変更される。
3. 一般には、REPプレフィックスはこの命令で用いられない。
4. 他の8086の操作と同じく、この命令の汎用形式は、8あるいは16ビットのどちらの操作が行なわれるかを決めるために、アセンブラに何らかの記号が与えられることを必要とする。この問題についてはこの章の後で述べる。

LOOP disp (Loop)

CXレジスタをデクリメントし、0 でなければジャンプする。

この命令は、CXレジスタをデクリメントし（フラグに影響は与えない）、減少によってCXレジスタが0にならなければジャンプを行ない、それ以外では次の命令を実行することを除けば、JMP disp 命令と同様の機能を行なう。

命令コードを次に示す。



例として、次の一連の命令を考える。

```

MOV     CX, LENGTH$OF$PAYROLL$ARRAY
PAYROLL$SUMMATION$ARRAY:
    } 給料支払い総額の計算
    LOOP PAYROLL$SUMMATION$ARRAY
  
```

PAYROLL\$SUMMATION\$ARRAY と LOOP 命令の間の一連の命令が、LENGTH\$OF\$PAYROLL\$ARRAY 回実行される。

サイクル数：分岐したとき：17

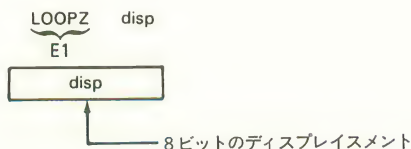
分岐しなかったとき：5

LOOPZ disp (Loop if Zero)
 LOOPE disp (Loop if Equal)

CXレジスタをデクリメントし、CXが0でなくZF=1ならば、ジャンプする。

この命令は、CXレジスタをデクリメントし（フラグに影響は与えない）、減少によってCXレジスタが0にならずしかもゼロ・フラグが1ならばジャンプを行ない、それ以外では次の命令を実行することを除けば、JMP disp 命令と同様の機能を行なう。

命令コードを次に示す。



例として、次の一連の命令を考える。

```

MOV      CX, NUMBER$OF$PORTS
MOV      DX, MAIN$PORT$GROUP
MOV      BX, REDUNDANT$PORT$GROUP

TOP:     IN      AX, DX
         INC     DX
         XCHG    BX, DX
         XCHG    AX, BP

         IN      AX, DX
         INC     DX
         XCHG    BX, DX
         CMP     AX, BP
         LOOPE   TOP
         JNZ     PORT$DISPUTE
  
```

TOPとLOOPE命令の間の一連の命令は、2つの入力ポートからのデータを比較する。1つのグループはMAIN\$PORT\$GROUPで示され、もう1つのグループはREDUNDANT\$PORT\$GROUPで示されている。次の2つの場合の1つが発生すれば、命令JNZ PORT\$DISPUTEが実行される。

1. 比較の結果、ゼロ・フラグが0に設定される。これはポートのデータが等しくない場合である。
2. TOPとLOOPE命令の間の命令が、NUMBER\$OF\$PORTS回実行された場合。JNZ命令は、1と2の場合を区別するために用いられる。

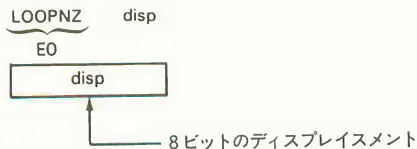
サイクル数：分岐したとき：18
 分岐しなかったとき：6

LOOPNZ disp (Loop if Not Zero)
LOOPNE disp (Loop if Not Equal)

CXレジスタをデクリメントし、CXが0でなくZF=0ならばジャンプする。

この命令は、CXレジスタをデクリメントし（フラグに影響は与えない）、減少によってCXレジスタが0にならずしかもゼロ・フラグが0ならばジャンプを行ない、それ以外では次の命令を実行することを除けば、JMP disp 命令と同様の機能を行なう。

命令コードを次に示す。



例として、次の一連の命令を考える。

```

MOV     SI,ELEMENT$TO$MATCH
LES     DI
MOV     CX,NUMBER OF ENTRIES

SEARCH$FOR$MATCH:
-
-                               ;SEARCH FOR MATCH
-
LOOPNE  SEARCH$FOR$MATCH
  
```

SEARCH\$FOR\$MATCH と LOOPNE 命令の間のコードは、

- 1) CXが減少して0になる。あるいは、
- 2) LOOPNEの前の命令がゼロ・フラグを1にする。たとえばマッチしたときにゼロ・フラグを1にするまで、繰り返し実行される。

サイクル数：分岐したとき：19
 分岐しないとき：5

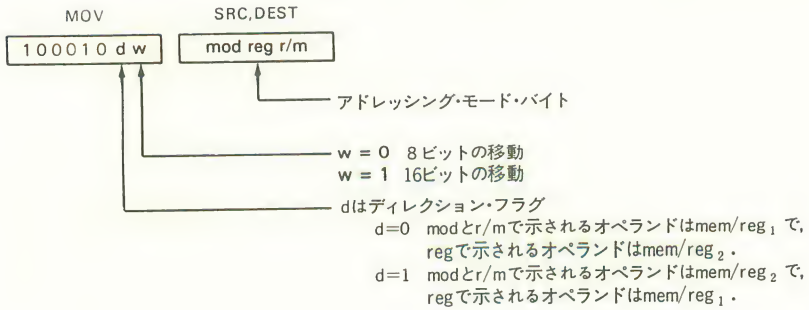
MOV mem/reg1, mem/reg2 (Move)

$\left\{ \begin{array}{l} \text{レジスタからレジスタへ} \\ \text{メモリからレジスタへ} \\ \text{レジスタからメモリへ} \end{array} \right\}$ データを移動する。

この命令は、レジスタとレジスタあるいはメモリの間で、8あるいは16ビットのデータ

要素の移動に用いられる。

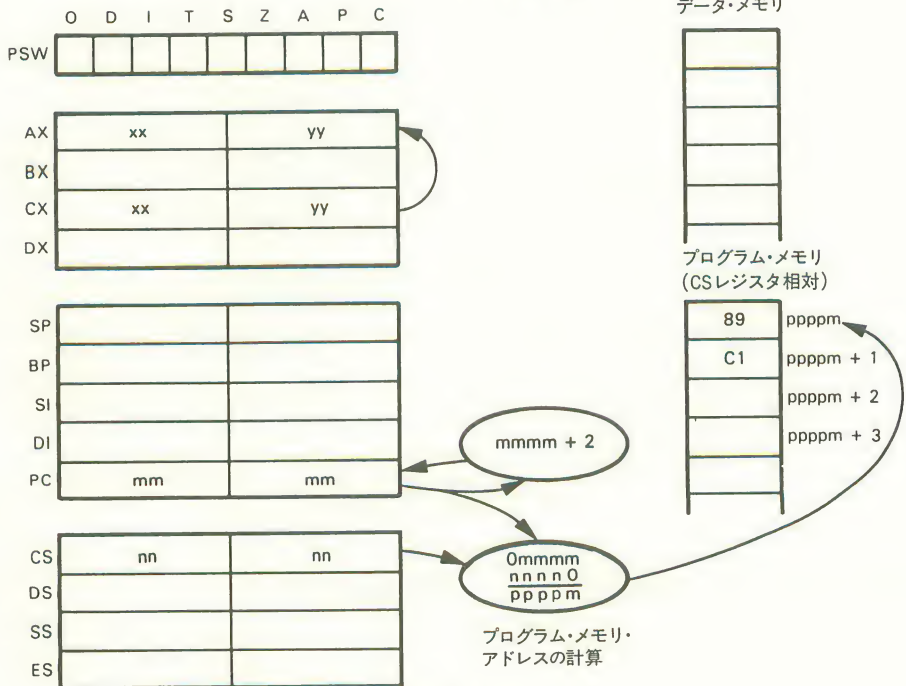
命令コードを次に示す。



たとえば、

MOV AX, CX

の命令は、CXレジスタの内容をAXレジスタに移動する。



MOV AX, CX

サイクル数:

レジスタからレジスタ: 2

メモリからレジスタ: 8+EA

レジスタからメモリ: 9+EA

注)

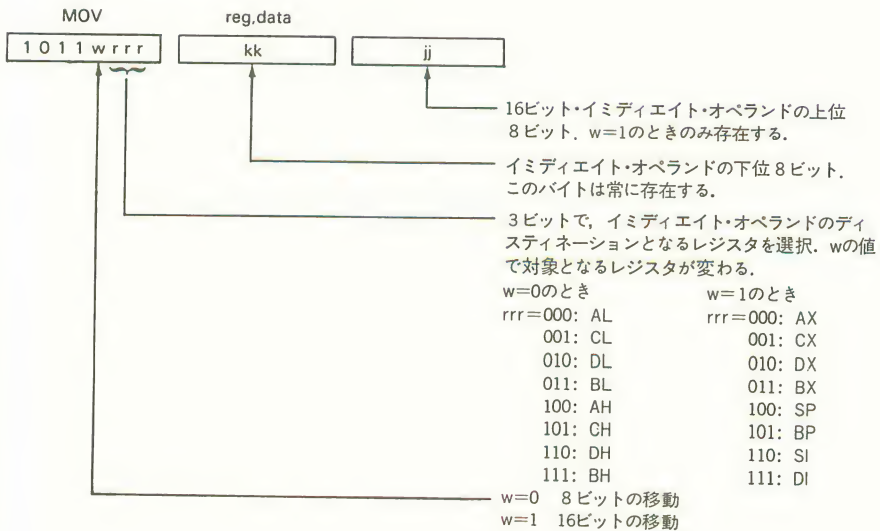
1. この命令では、セグメント・レジスタを指定できない。セグメント・レジスタのデータ移動については、MOV segreg, reg あるいは MOV reg, segreg の命令を参照。
2. ステータスは影響を受けない。
3. この命令は、8080アセンブリ命令の MOV reg, reg 命令で行なわれる機能を果たす。しかし、この命令は、対応する8080命令よりも融通性のある使い方ができる。

MOV reg, data (Move)

レジスタにイミディエイト・データをロードする。

この命令は、イミディエイト・アドレッシングによって、レジスタに8あるいは16ビットのデータ要素をロードするのに用いられる。

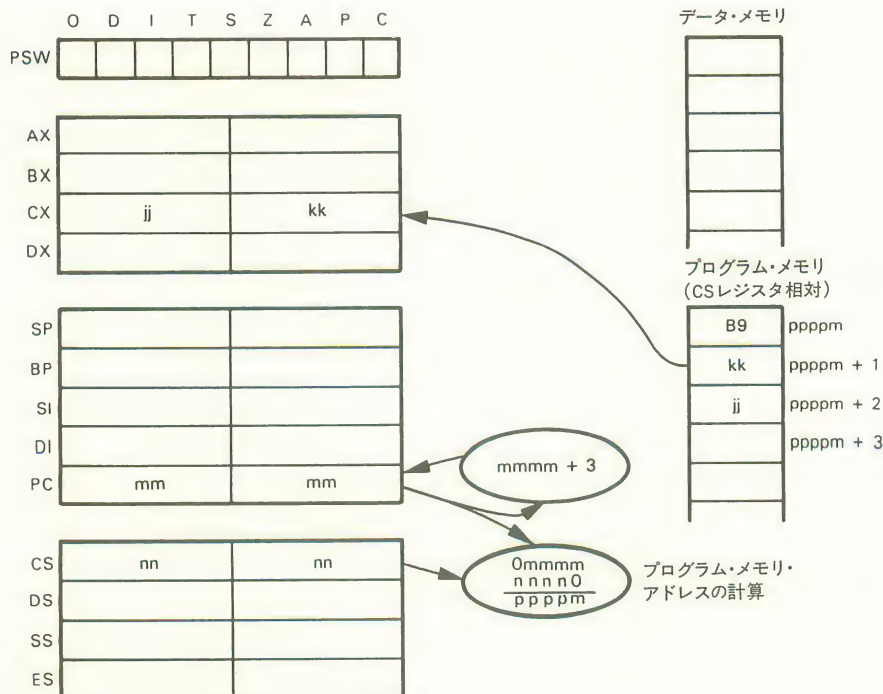
命令コードを次に示す。



例として、

MOV CX, 3168H

の命令は、CXレジスタに16ビットの値 3168₁₆ を移動する。



MOV CX, jkk
サイクル数: 4

注)

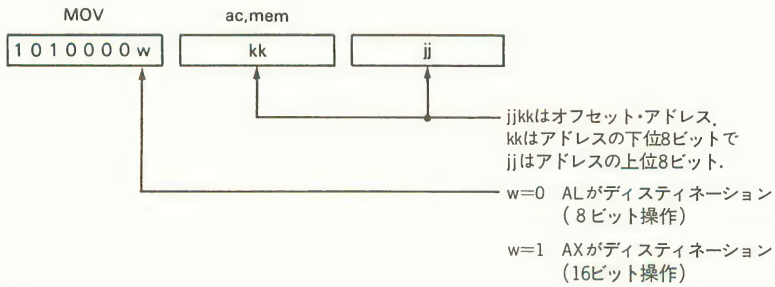
1. この命令で、セグメント・レジスタにロードはできない。セグメント・レジスタにイミディエイト・データをロードするには、MOV segreg,mem/reg 命令を参照。
2. この命令は、8080で実行されるMVI (8ビットのイミディエイトの移動) とLXI (16ビットのイミディエイトの移動) の命令の機能を果たす。
3. ステータスは影響を受けない。

MOV ac, mem (Move)

メモリからアキュムレータにロードする。

この命令は、メモリからアキュムレータに、8あるいは16ビットのデータ要素を移動するために用いられる。

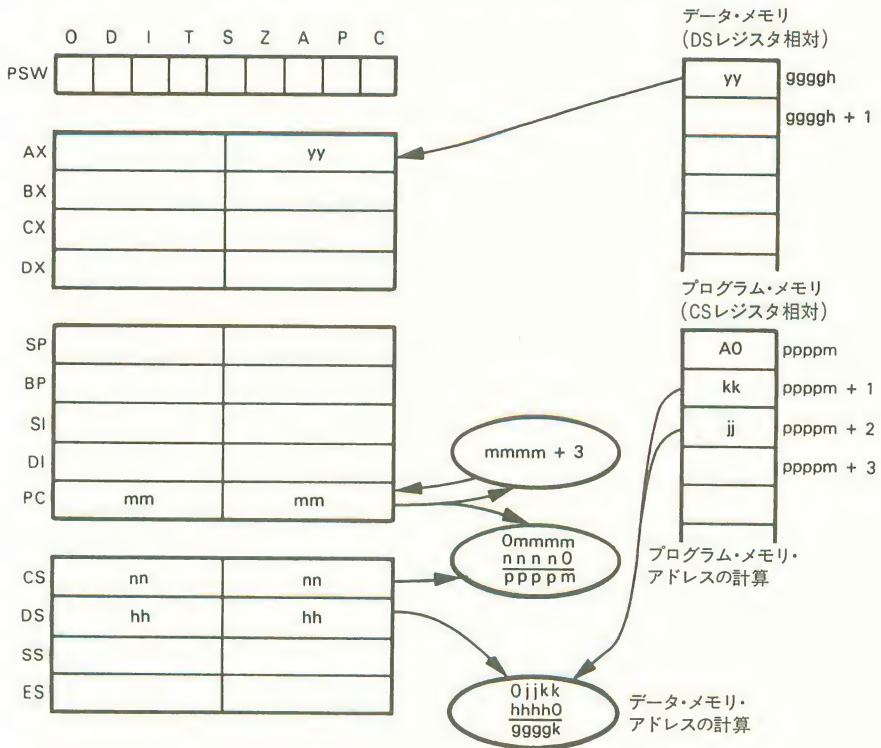
命令コードを次に示す。



たとえば、

MOV AL, [1064H]

の命令は、メモリ位置 1064_{16} (DSレジスタ相対) の内容をALレジスタに移動する。



注)

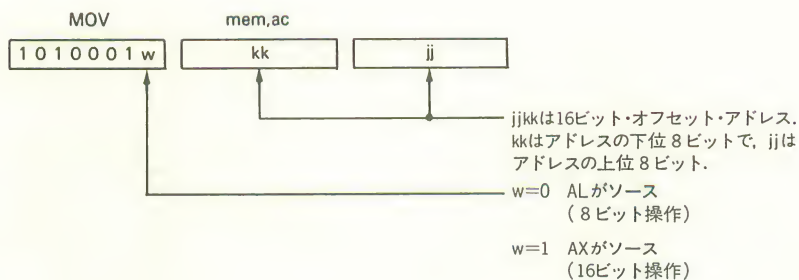
- この命令は、8080の命令 LDA addr と同じ機能を果たす。また、AXレジスタへの16ビットのロードもできる。

MOV mem, ac (Move)

アキュムレータからメモリにストアする。

この命令はアキュムレータからメモリに、8あるいは16ビットのデータ要素を移動するために用いられる。

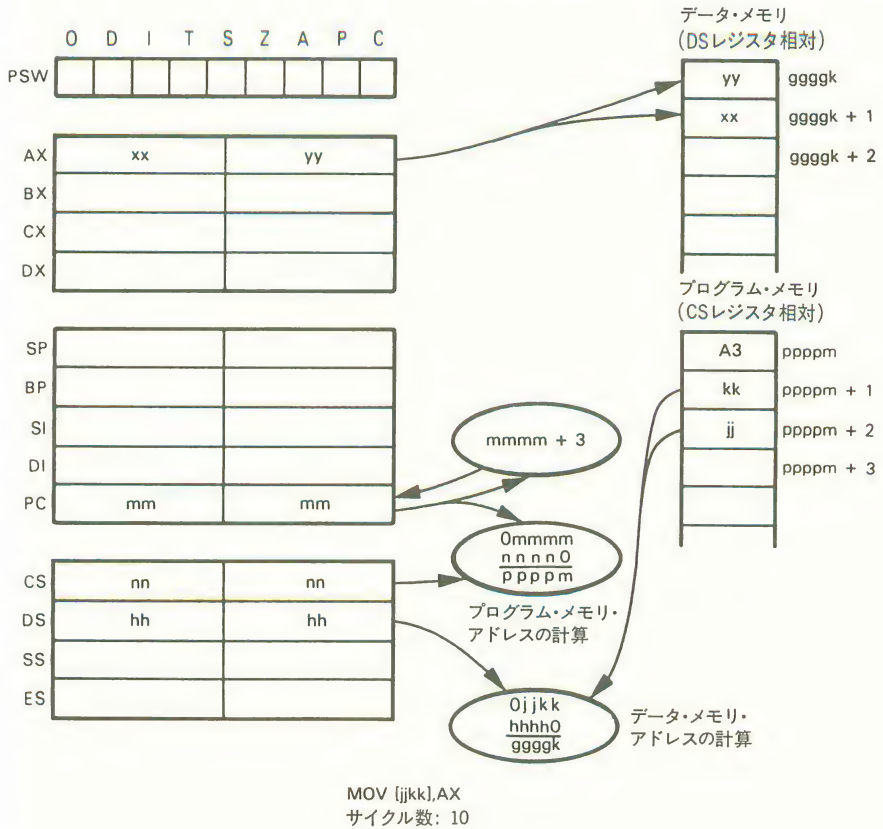
命令コードを次に示す。



たとえば、

MOV [1064H], AX

の命令は、AXレジスタの内容をメモリ位置 1064_{16} (DSレジスタ相対) に移動する。ALレジスタの内容は 1064_{16} に移動し、AHレジスタの内容は 1065_{16} に移動する。



注)

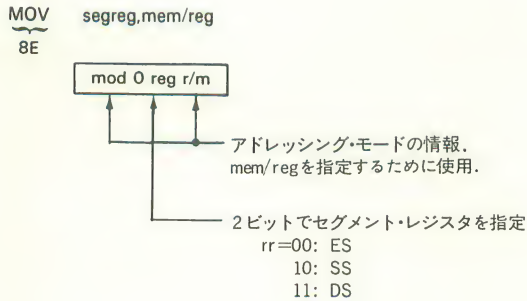
1. ステータスは影響を受けない。
2. この命令は、8080の命令 STA·addr と同じ機能を果たす。さらに、AXレジスタの16ビットのストアができる。

MOV segreg, mem/reg (Move)

メモリあるいはレジスタのデータをセグメント・レジスタに移動する。

レジスタあるいはメモリから、16ビットのデータ要素をセグメント・レジスタに移動する。

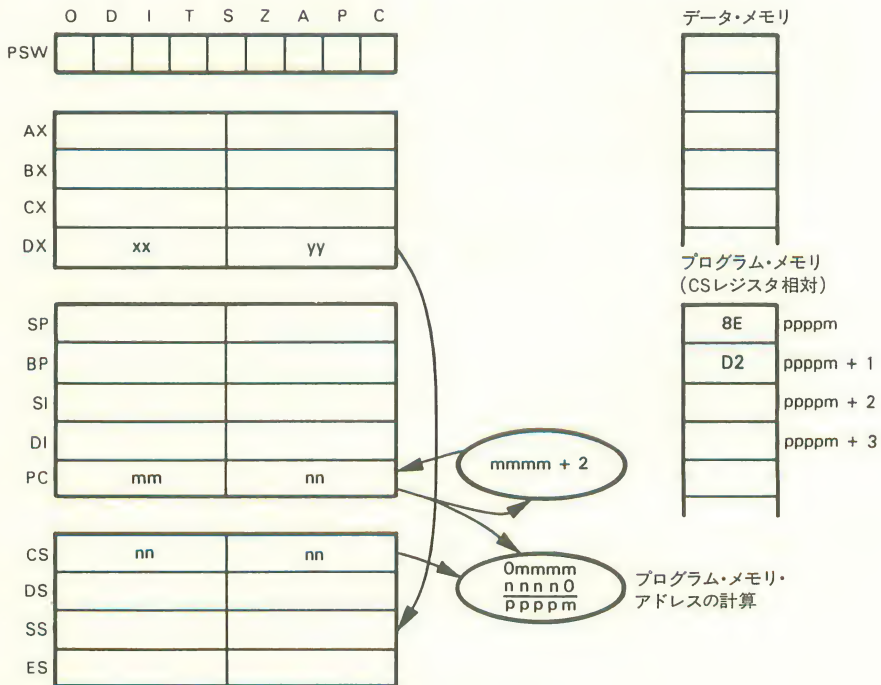
命令コードを次に示す。



例として、

MOV SS,DX

の命令によって、DXレジスタの内容がSSレジスタに移動する。



MOV SS,DX

サイクル数: レジスタからレジスタ: 2

メモリからレジスタ: 8+EA

注)

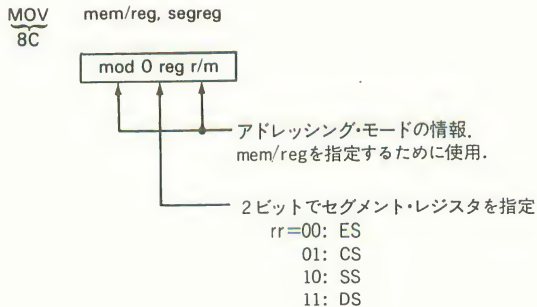
1. $\text{reg}=01$ の場合、この命令の結果は定義されない。この制限によって、ユーザがCSレジスタに直接ストアすることを防止している。CSレジスタに対する変更は、PCへのロードも行なうJMP, CALL, RET, IRET, INTの命令によってのみ行なわれる。
2. この命令は、一般にプログラムのセグメント領域が定義される初期設定で用いられる。
3. インタラプトは、この命令の終わりではサンプルされない。これに続く命令の終わりにサンプルされる。この制限は、インタラプトを起こさせずに、32ビットのポインタ全体の回復を可能にする。これは、SSとSPをロードする際に問題となる。

MOV mem/reg, segreg (Move)

セグメント・レジスタの内容をレジスタあるいはメモリに移動する。

セグメント・レジスタから16ビットのデータ要素を、レジスタあるいはメモリに移動する。

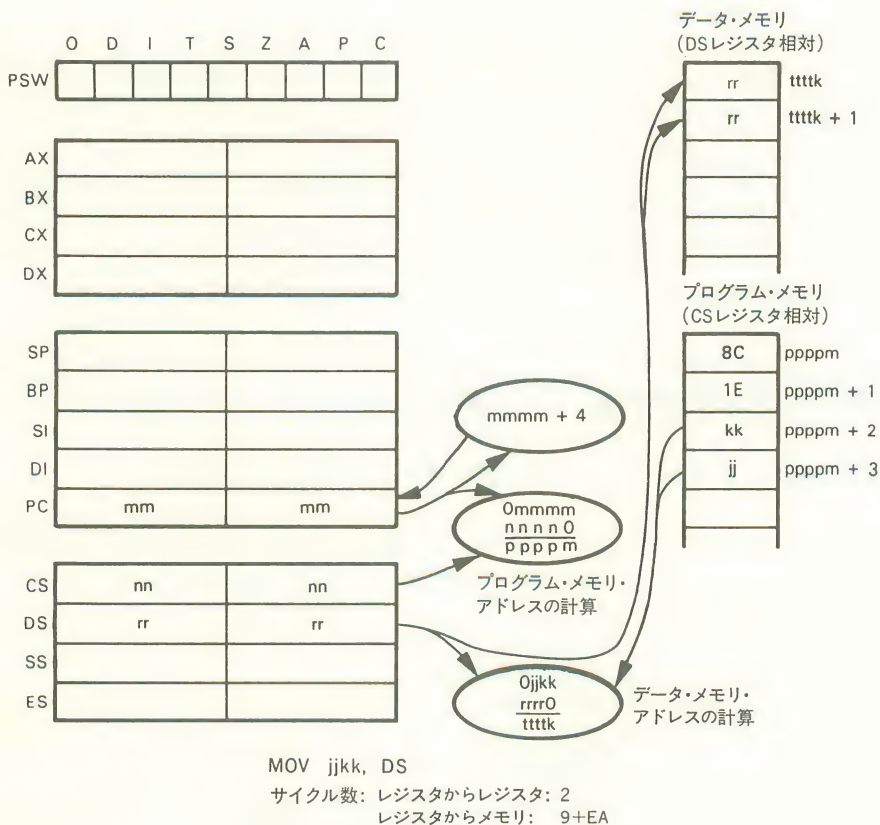
命令コードを次に示す。



たとえば、DSレジスタが 2000_{16} 含む場合を考える。

MOV 2000H, DS

の命令によって、メモリの 22000_{16} に 00_{16} がストアされ、 22001_{16} に 20_{16} がストアされる。



注)

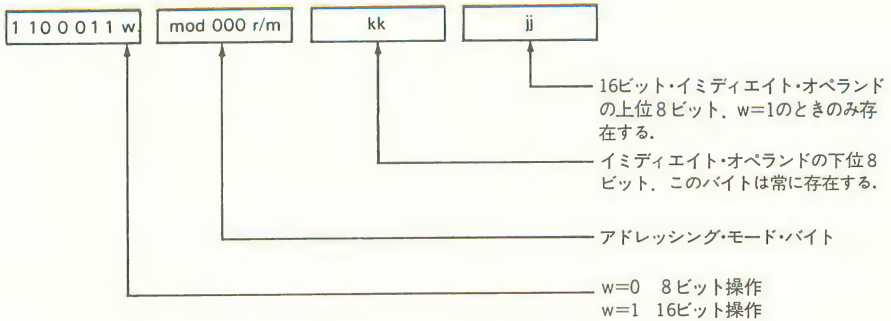
- これは、汎用レジスタ間のMOVではなく、セグメント・レジスタの移動用である。
汎用レジスタのMOVについては、MOV mem/reg1,mem/reg2 を参照。

MOV mem/reg,data (Move)

イミディエイト・データをレジスタあるいはメモリに移動する。

オペ・コードに続くバイトのイミディエイト・データを、指定されたレジスタあるいはメモリ位置に移動する。

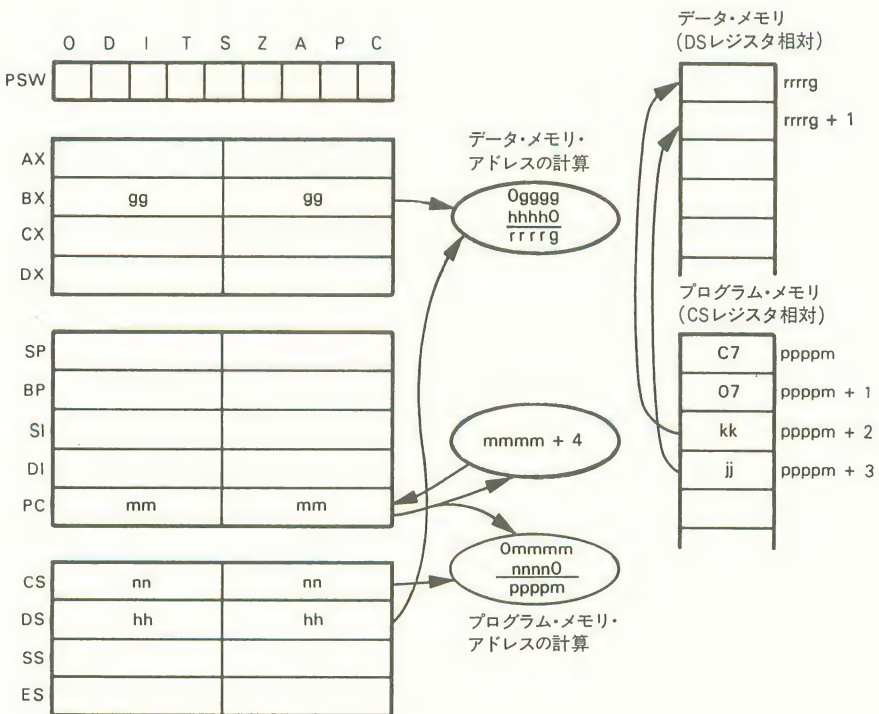
命令コードを次に示す。



たとえば、DSレジスタがD000₁₆を含み、BXレジスタが0016₁₆を含む場合を考える。

MOV [BX], 491FH

の実行後、メモリ位置 D0016₁₆は1F₁₆に、メモリ位置 D0017₁₆は49₁₆になる。



MOV [BX], jkk
サイクル数: 10+EA

注)

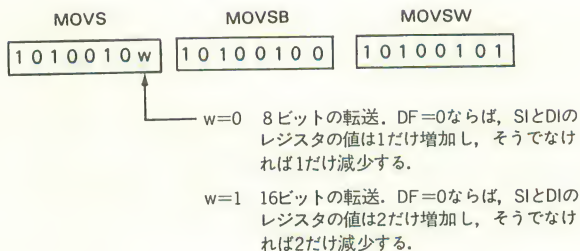
1. ステータスは影響を受けない。
2. セグメント・レジスタは、この命令で指定できない。
3. この命令は一般に、イミディエイト・データをレジスタに移動するためには用いられない。そのためには、MOV reg,data の命令がある。

MOVS/MOVS B/MOVS W (Move String)

バイトあるいはワードをメモリからメモリへ移動する。

8あるいは16ビットを、S Iレジスタで示されるメモリ位置からD Iレジスタで示されるメモリ位置に移動する。S IとD Iのレジスタは、DFフラグの値によって、増加あるいは減少する。

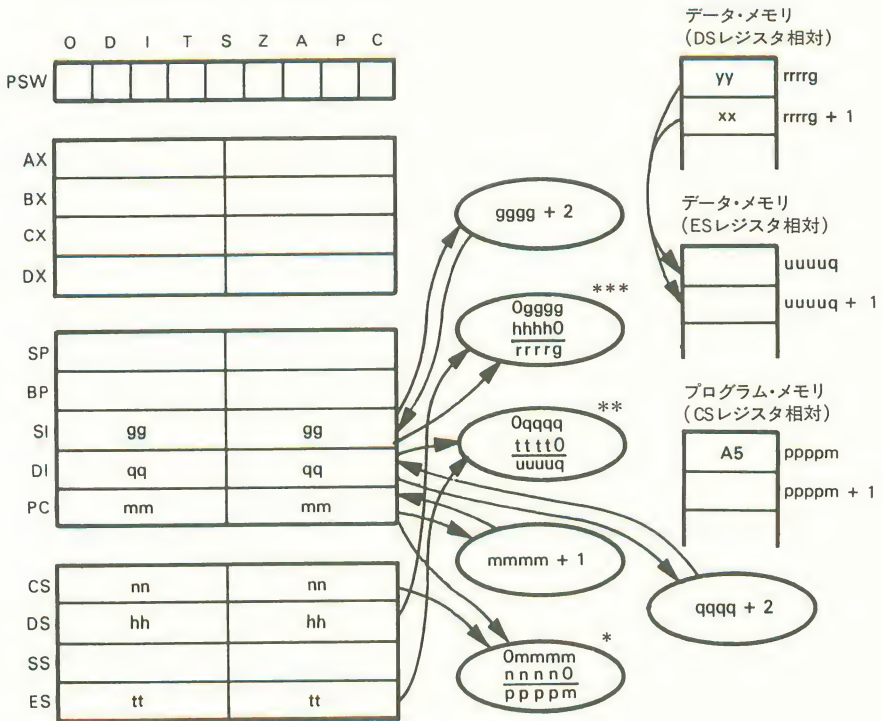
命令コードを次に示す。



DFフラグが0で、DSレジスタが1000₁₆を含み、ESレジスタが1780₁₆を含み、SIレジスタが0006₁₆を含み、DIレジスタが0006₁₆を含み、メモリ位置10006₁₆のワードが8F0B₁₆である場合を考える。

MOVS W

の実行後、メモリ位置17806₁₆は8F0B₁₆で、SIレジスタは0008₁₆に、DIレジスタは0008₁₆になる。



- * プログラム・メモリ・アドレスの計算
 ** ディスティネーション・データ・メモリ・アドレスの計算
 *** ソース・データ・メモリ・アドレスの計算

MOVSW

サイクル数: 18: 1度だけ実行のとき

9+(17*R): REPプレフィックスによってR回実行されたとき

注)

1. ステータスは影響を受けない。
2. ソース・オペランドのデフォルト・セグメント・レジスタはDSレジスタである。このセグメントは、セグメント・プレフィックスを用いて変えられる。ディスティネーション・オペランドのデフォルト・セグメント・レジスタはESレジスタである。このセグメントは、セグメント・プレフィックスによって変えられない。
3. REPプレフィックスやLOCKプレフィックスはこの命令と共に用いられる。この命令と共に、REPとLOCKのプレフィックスが用いられた場合は問題となる。この問題については次章を参照。
4. この命令は、メモリ・ブロックの移動に便利である。次の一連の命令を考える。

```

LES      DI, CURRENT$START$OF$PRINT$BUFFER
MOV      SI, PAGE$HEADER$MESSAGE
MOV      CX, PAGE$HEADER$MESSAGE$LENGTH
REP
MOVS     BYTE

```

これは、PAGE\$HEADER\$MESSAGEで示されるメモリ位置のデータを、CURRENT\$START\$OF\$PRINT\$BUFFERの内容で示されるメモリ位置に移動する。

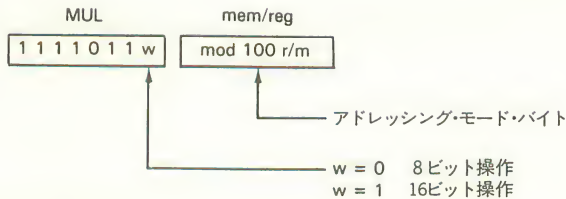
5. MOV Sの汎用の形式に対して、8あるいは16ビットの移動のどちらをどのように指定するかは、用いるアセンブラに依存している。この件についての解説は、この章の終わりを参照のこと。

MUL mem/reg (Multiply)

ALあるいはAXレジスタにレジスタあるいはメモリを乗じる。

2つのオペランドを符号なし数値、すなわち単純な2進数と見なして、指定されたレジスタあるいはメモリの内容と、AL（8ビット操作）あるいはAX（16ビット操作）のレジスタとの乗算を行なう。8ビット操作の場合、結果の下位8ビットはALレジスタにストアされ、結果の上位8ビットはAHレジスタにストアされる。16ビット操作では、結果の下位16ビットはAXレジスタにストアされ、結果の上位16ビットはDXレジスタにストアされる。どちらの場合も、結果の上位1/2が0ならば、OFとCFは0となり、そうでなければAHあるいはDXに有効数字があることを示すために、OFとCFは1になる。

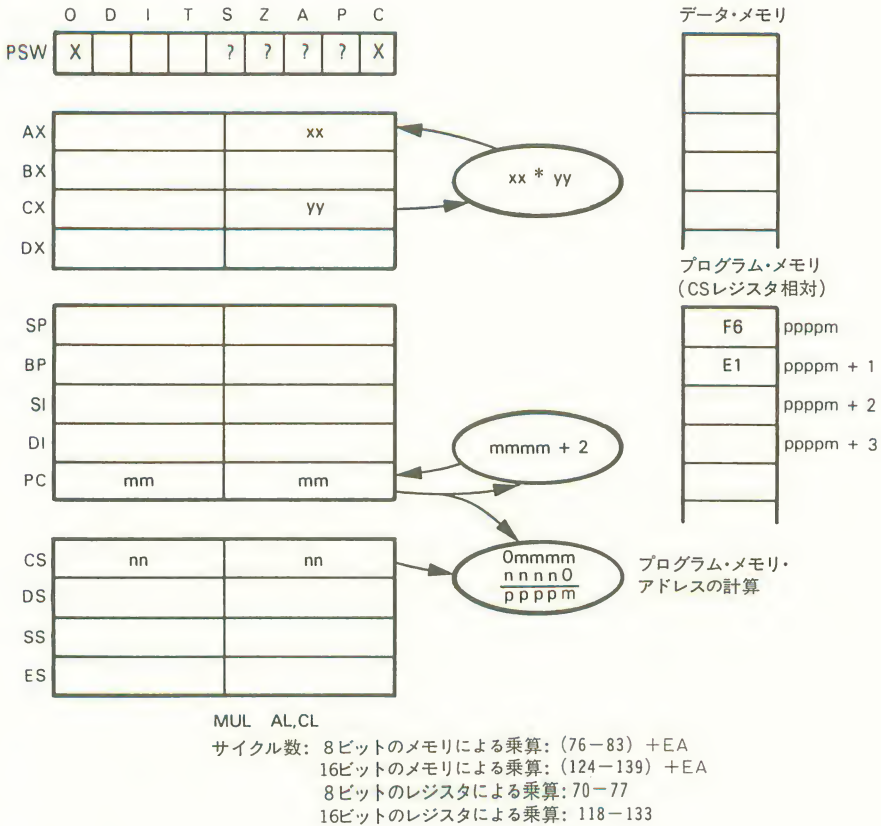
命令コードを次に示す。



例として、AXレジスタが4514₁₆を含み、CLレジスタが05₁₆を含む場合を考える。

MUL AL, CL

の実行後、AXレジスタは0064₁₆になり、キャリーとオーバーフローのフラグは0となる。



注)

- これは符号なしの乗算である。2つのオペランドは、以下の範囲の符号なし2進数として取り扱われる。

8ビット: 0~255

16ビット: 0~65535

符号付きの乗算については、命令 `IMUL` を参照。

- ある場合には、乗算を行なうためにシフトを用いることがより適当となるときがある。このような場合は、メモリの保存が最も重要ではなく速さが必要なときに生じる。

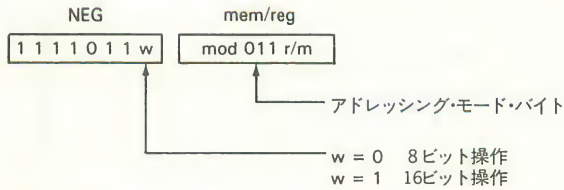
NEG mem/reg (Negate)

レジスタあるいはメモリの内容の補数をとる。

この命令は、0から指定されたオペランドの2の補数による減算を行なう。結果は指定

されたオペランドにストアされる。8あるいは16ビットのオペランドが指定できる。

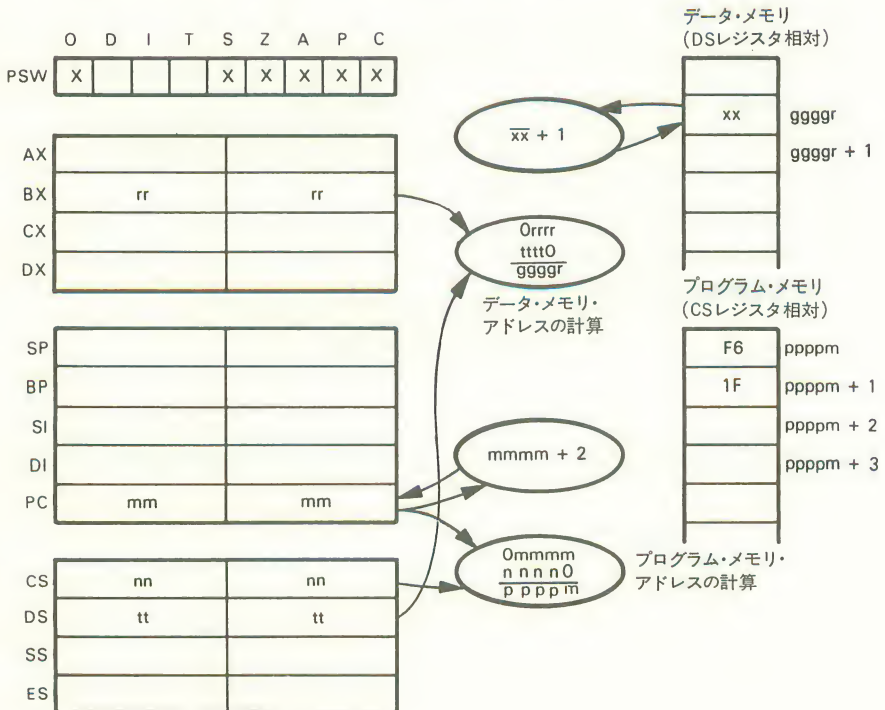
命令コードを次に示す。



B Xレジスタが 0006_{16} を含み、D Sレジスタが 1800_{16} を含み、メモリ位置 18006_{16} の内容が 47_{16} であるとする。

NEG [BX]

の命令実行後、メモリ位置 18006_{16} の内容は $B 9_{16}$ になる。



NEG [BX]

サイクル数: メモリ・オペランド: 16+EA
レジスタ・オペランド: 3

注)

1. 8080アセンブリ言語には、等価な命令は存在しない。16ビットの数値に対してこの命

令と等価な8080の処理は次のようになる。

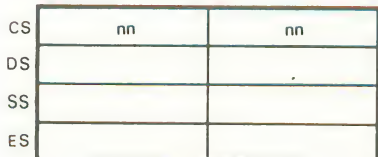
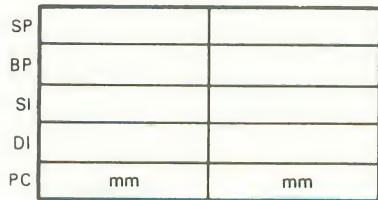
MOV	A, D
GMA	
MOV	D, A
MOV	A, E
GMA	
MOV	E, A
INX	D

NOP (No Operation)

無操作 (ノー・オペレーション)。

何も実行しない。命令コードを次に示す。

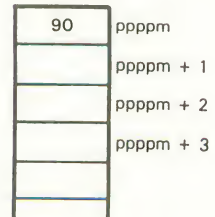
NOP
90



データ・メモリ



プログラム・メモリ
(CSレジスタ)



mmmm + 1

0mmmm
nnnnn0
ppppm

プログラム・メモリ・
アドレスの計算

NOP

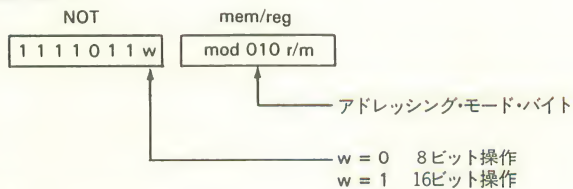
サイクル数: 3

NOT mem/reg (NOT)

レジスタあるいはメモリの1の補数をとる。

指定されたレジスタあるいはメモリ位置の内容の補数をとる。

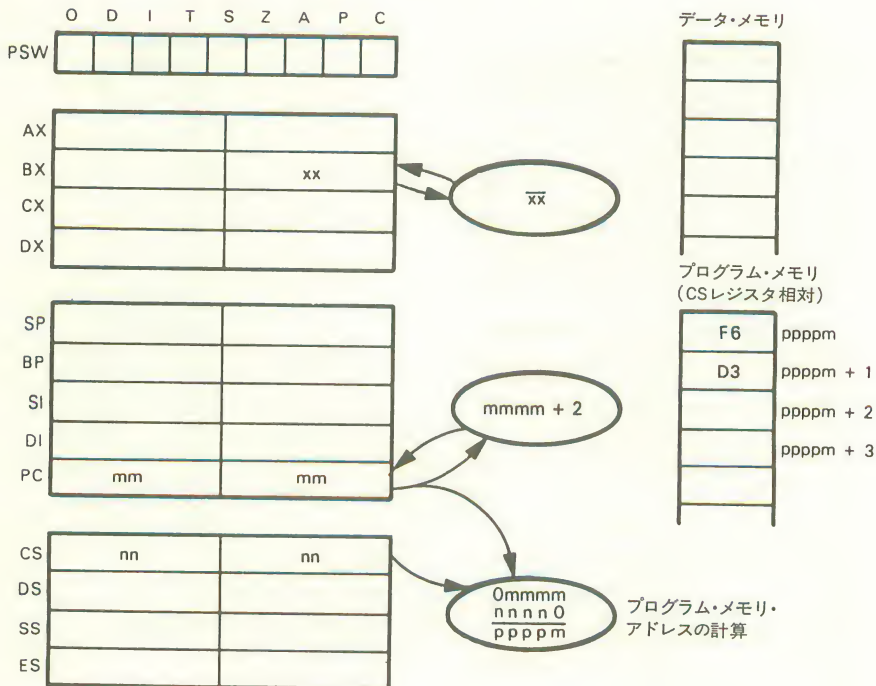
命令コードを次に示す。



B LレジスタがFB₁₆を含むとする。

NOT BL

の実行後、B Lレジスタは04₁₆になる。



NOT BL

サイクル数: メモリ・オペランド: 16+EA
レジスタ・オペランド: 3

注)

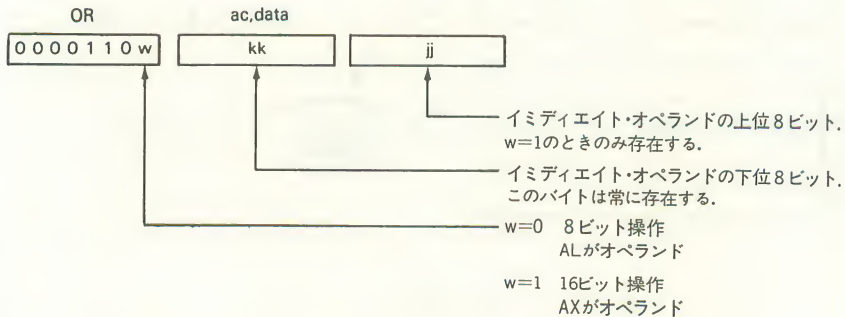
1. ステータスは影響を受けない。
2. この命令は、8080の命令 CMA と同じ機能を果たす。この命令は16ビットの補数をとることもでき、任意の汎用レジスタあるいはメモリ位置も指定できる。

OR ac,data (OR)

イミディエイト・データとAXあるいはALレジスタのORをとる。

後続のプログラム・メモリ・バイトのイミディエイト・データと、AL（8ビット操作）あるいはAX（16ビット操作）のレジスタのORをとる。

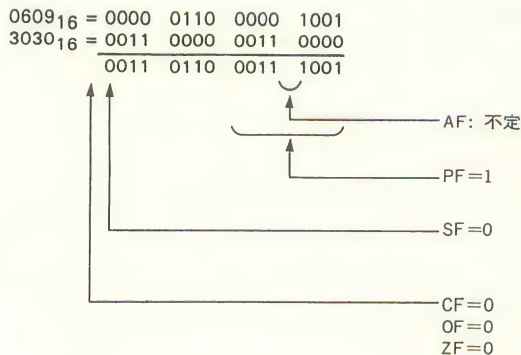
命令コードを次に示す。

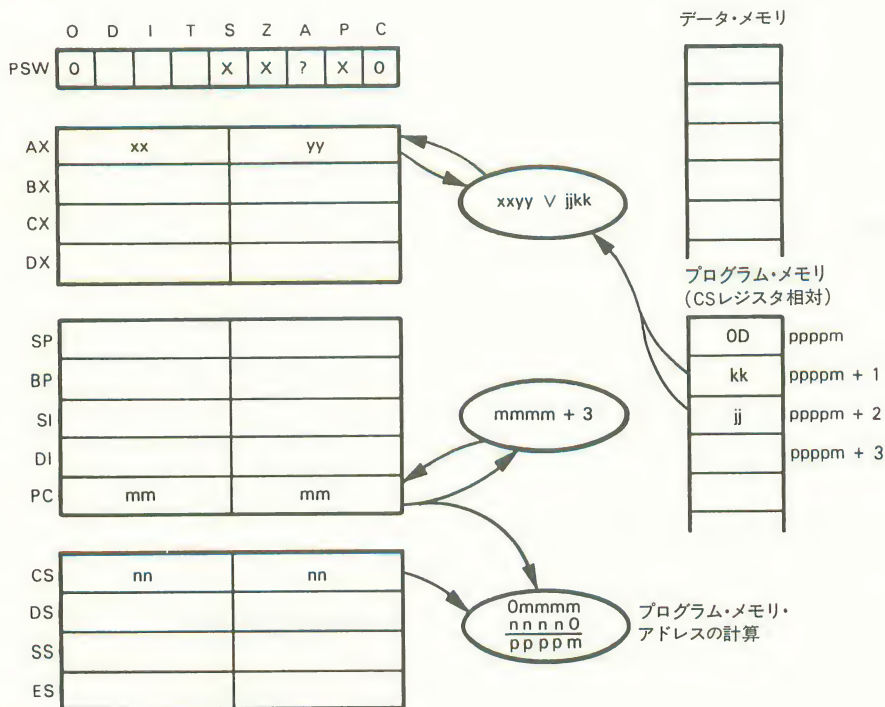


AXレジスタが0609₁₆を含むと仮定する。

OR AX,3030H

の実行後、AXレジスタは3639₁₆になる。





OR AX,jkk
サイクル数: 4

注)

1. この命令は、8080の命令 `ORI data` と同じ機能を果たす。また、16ビットの操作を行なうこともできる。
2. イミディエイト・データと、他の汎用レジスタあるいはメモリとのORが必要ならば、`OR mem/reg,data` の命令を参照。

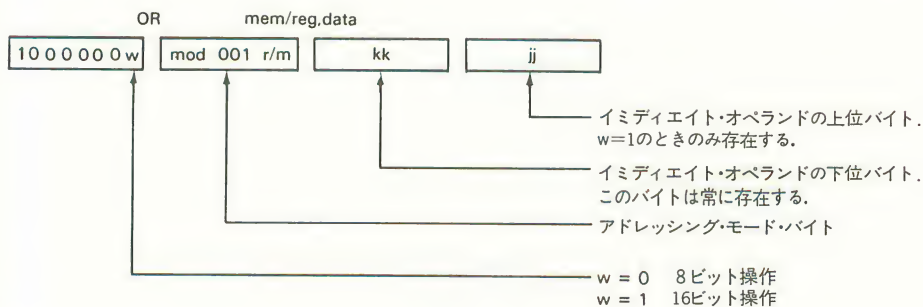
OR mem/reg,data (OR)

イミディエイト・データとレジスタあるいはメモリのORをとる。

後続のプログラム・メモリ・バイトのイミディエイト・データと、指定されたレジスタあるいはメモリ位置とのORをとる。

8あるいは16ビットの操作が指定できる。

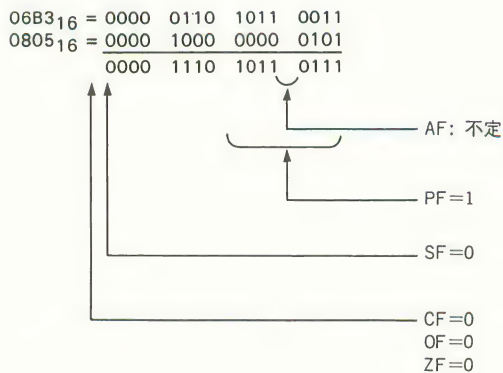
命令コードを次に示す。

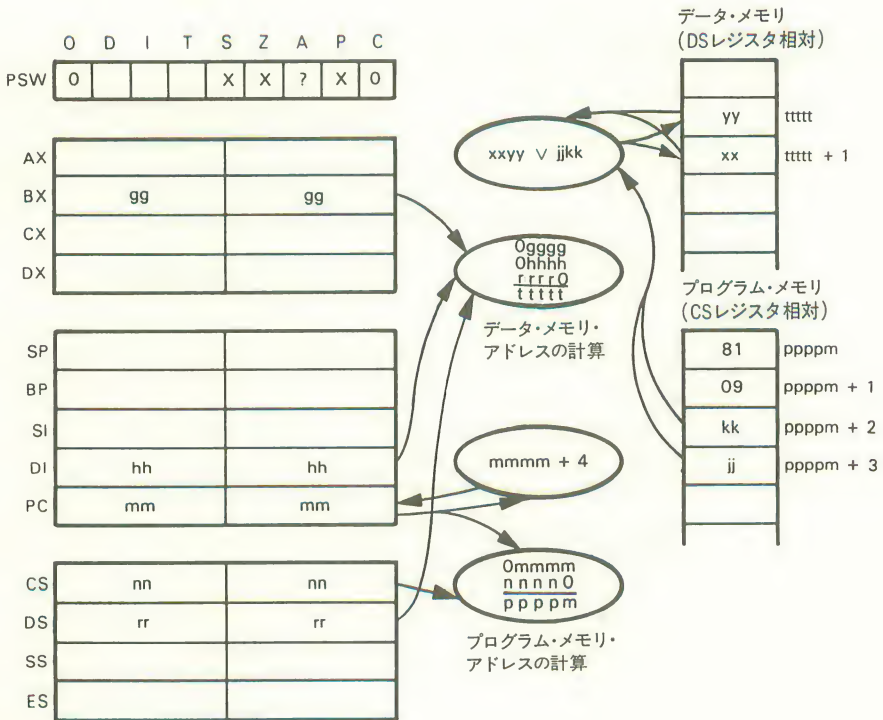


D S レジスタが 3800_{16} を含み, B X レジスタの内容が 0200_{16} で, D I レジスタが 0136_{16} を含み, メモリ位置 38336_{16} のワードが $06B3_{16}$ であるとする.

OR [BX+DI], 0805H

の実行後, メモリ位置 38336_{16} のワードは $0EB7_{16}$ になる.





注)

1. この命令は、一般にイミディエイト・データとAXあるいはALレジスタのORをとるためには用いられない。このためには、OR ac,data の命令がある。

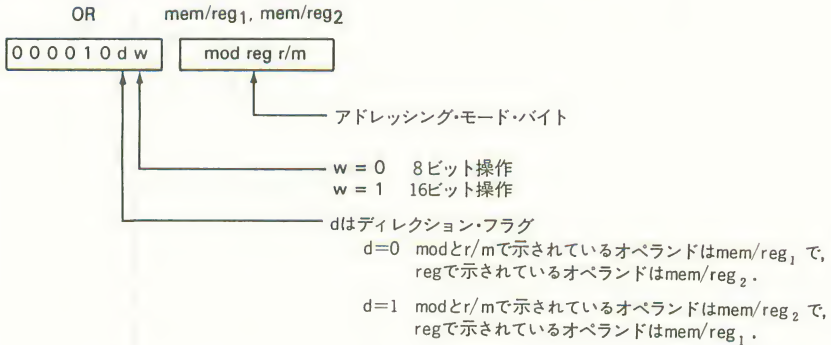
OR mem/reg₁, mem/reg₂ (OR)

$\left\{ \begin{array}{l} \text{レジスタとレジスタ} \\ \text{レジスタとメモリ} \\ \text{メモリとレジスタ} \end{array} \right\}$ のORをとる。

mem/reg₂で示されるレジスタあるいはメモリ位置の内容と、mem/reg₁で示されるレジスタあるいはメモリ位置の内容のORをとり、結果をmem/reg₁に返す。8あるいは16ビットの操作が指定できる。mem/reg₁あるいはmem/reg₂はメモリ・オペランドとなるが、オ

ペランドの一方はレジスタ・オペランドでなければならない。

命令コードを次に示す。

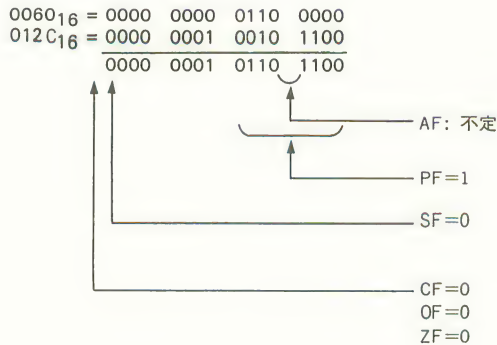


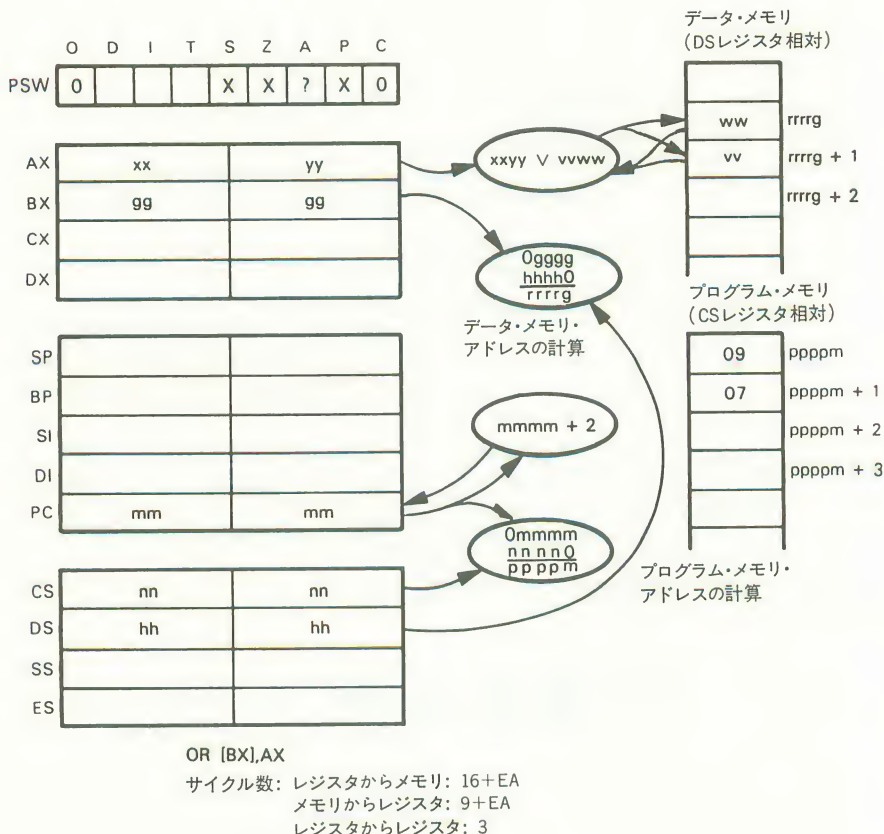
A Xレジスタが0060₁₆を含み、D Sレジスタが4000₁₆を含み、B Xレジスタが009A₁₆を含み、メモリ位置4009A₁₆のワードが012C₁₆であるとする。

OR [BX], AX

の実行後、メモリ位置4009A₁₆のワードは016C₁₆になる。

フラグは次のように設定される。



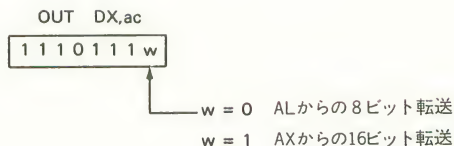


OUT DX,ac (Output)

アキュムレータの内容を出力する。

AL (8ビット) あるいは AX (16ビット) のレジスタの 8 あるいは 16ビットのデータ要素を、DX レジスタの内容で指定される I/O ポートに出力する。

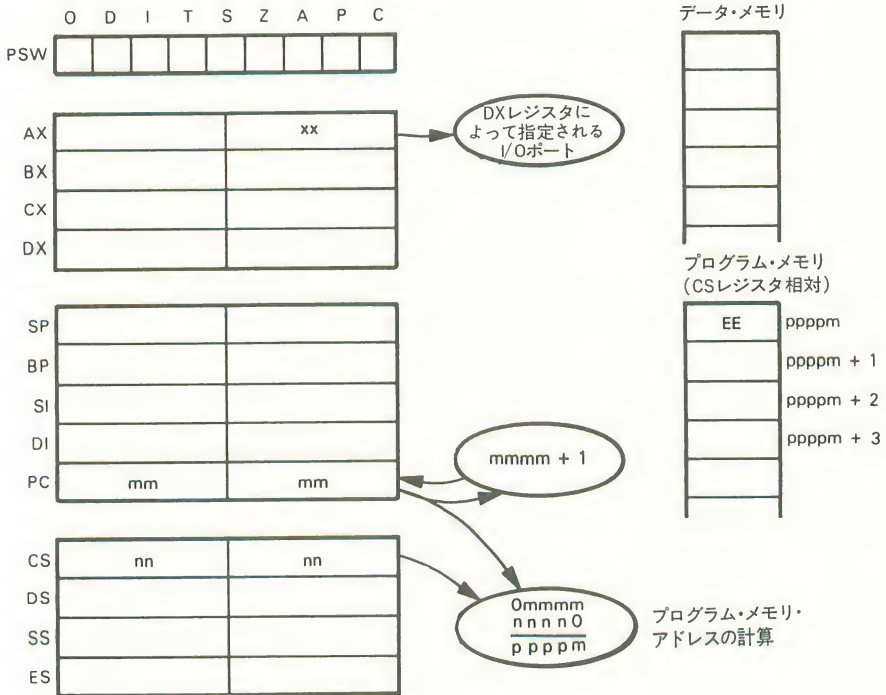
命令コードを次に示す。



例として、DXレジスタが FFF2_{16} を含み、ALレジスタが 40_{16} の場合を考える。

OUT DX,AL

の実行によって、ポート番号 FFF2_{16} のI/Oポートのバッファに、 40_{16} がロードされる。



OUT DX,AL

サイクル数: 8

注)

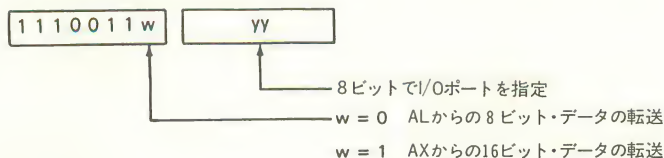
1. この命令によって、0 から FFFF_{16} までの番号で指定されるI/Oポートにアクセスできる。
2. レジスタあるいはステータスは影響を受けない。

OUT port,ac (Output)

アキュムレータの内容を出力する。

この命令は、AL (8ビット) あるいはAX (16ビット) のレジスタの8あるいは16ビットのデータ要素を、命令の2番目のバイトで指定されるI/Oポートに出力する。

命令コードを次に示す。

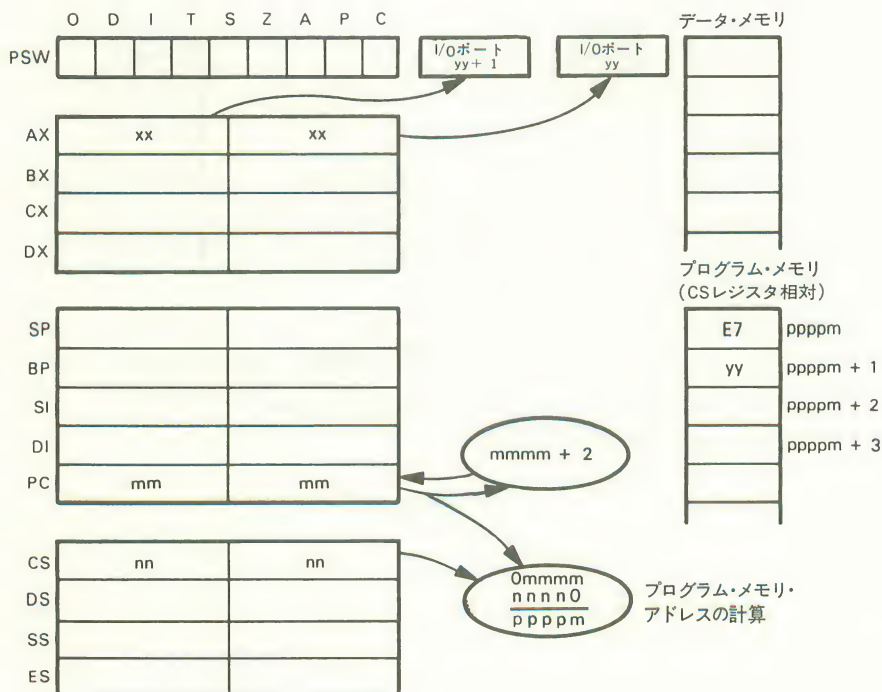


レジスタあるいはフラグは影響を受けない。

A Xレジスタが $58A4_{16}$ を含むとする。

OUT 14H,AX

の実行によって、 14_{16} のI/Oポートに $A4_{16}$ が、 15_{16} のI/Oポートに 58_{16} が転送される。



OUT yy,AX

サイクル数: 10

注)

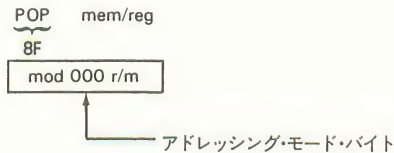
- この命令によって、0 から FF_{16} までの番号で指定されるI/Oポートにアクセスできる。この範囲外のポートについては、OUT DX,ac の命令を参照。
- この命令は、8080の命令 OUT port と同じ機能を果たす。さらに1個の命令で16ビット：データの転送が可能である (8080の命令 OUT port では不可能)。

3. OUT命令を有効に用いるためには、ハードウェアの構成を十分に理解することが必要である。I/O ロジックの構成方法によって、種々のハードウェアの機能にアクセスする際に用いられるポート・アドレスが決定される。特定のメモリ・アドレスでメモリ参照命令を用いることで外部ロジックにアクセスするマイクロコンピュータ・システムを設計することもできる*。

POP mem/reg (Pop)

スタックのトップからリードする。

スタックのトップの2バイトを、指定されたメモリ位置あるいはレジスタにポップする。命令コードを次に示す。

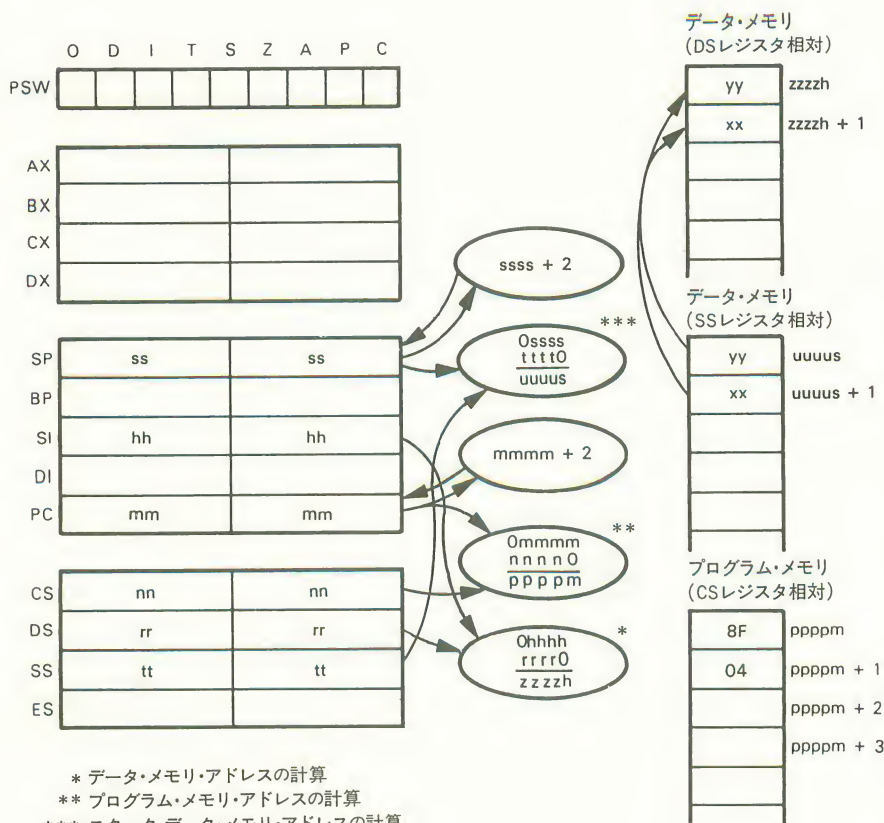


DSレジスタが $FF00_{16}$ を含み、SIレジスタが 0008_{16} を含み、SPレジスタが $0FEA_{16}$ を含み、SSレジスタが $2F00_{16}$ を含み、メモリ位置 $2F00_{16}$ のワードが $3CB5_{16}$ であると仮定する。

POP [SI]

の実行後、メモリ位置 $FF008_{16}$ の内容は $C5_{16}$ になり、メモリ位置 $FF009_{16}$ の内容は $3B_{16}$ になる。SPは $0FFC_{16}$ になる。

* メモリ・マップドI/O (Memory mapped I/O) (訳者注)。



POP [SI]

サイクル数: メモリ・オペランド: 17+EA

レジスタ・オペランド: 8

注)

- この命令は、一般にデータをレジスタにポップするためには用いられない。命令 PO P reg はこの機能を果たし、プログラム・メモリの1バイトしか占有しない。
- ステータスは影響を受けない。

POP reg (Pop)

スタックのトップからリードする。

スタックのトップの2バイトを、指定されたレジスタにポップする。
命令コードを次に示す。



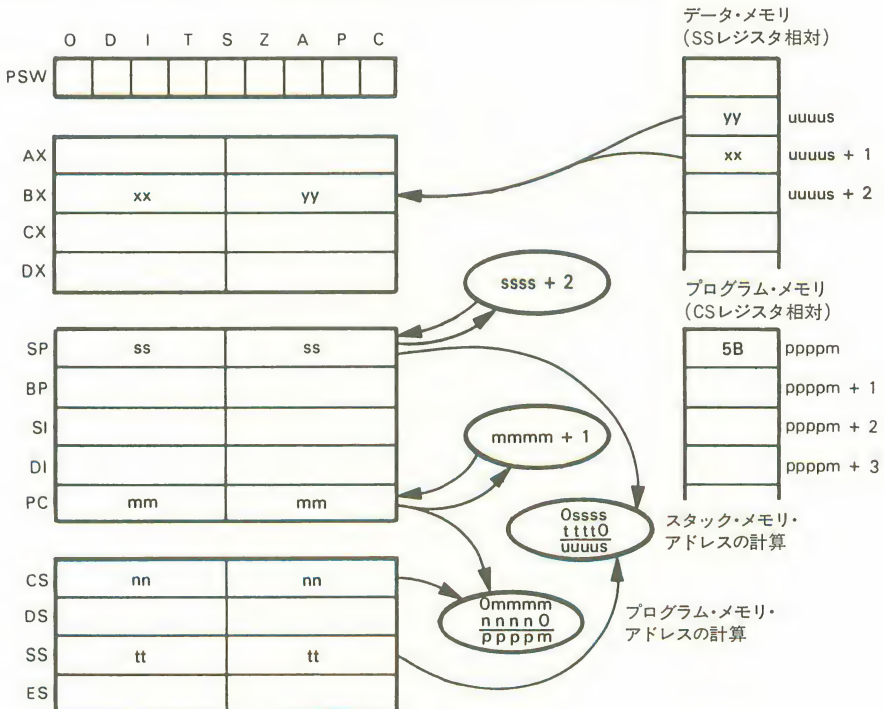
3ビットで、データがポップされる16ビット・レジスタを指定。

rrr=000: AX
 001: CX
 010: DX
 011: BX
 100: SP
 101: BP
 110: SI
 111: DI

たとえば、次の命令を考える。

POP BX

この命令は、スタック・ポインタ（スタック・セグメント）で示されるバイトをBLにポップし、スタック・ポインタをインクリメントしてそのとき示されるバイトをBHにポップする。最後に、再びスタック・ポインタを1だけ増加して、スタックの新しいトップを示すようにする。これで、実際8086における1つの16ビット転送が行なわれる。



サイクル数: 8

注)

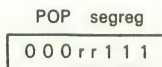
1. この命令は、セグメント・レジスタへのデータ要素のポップには使用できない。セグメント・レジスタへデータをポップするためには、命令 `POP segreg` を参照。
2. この命令が意味を持つためには、当然以下のことが必要となる。
 - a. スタック・ポインタが初期設定されている。
 - b. `PUSH` 命令によってスタックにデータが存在する。
 当然、`SP` レジスタを2だけ増加する目的でこの命令を用いることができるが、これは推奨できない。
3. この命令は、8080アセンブリ言語の命令 `POP reg` と同じ機能を果たす。

POP segreg (Pop)

スタックのトップからリードする。

この命令は、スタックのトップから2バイトを、指定された16ビットのセグメント・レジスタにポップする。

命令コードを次に示す。



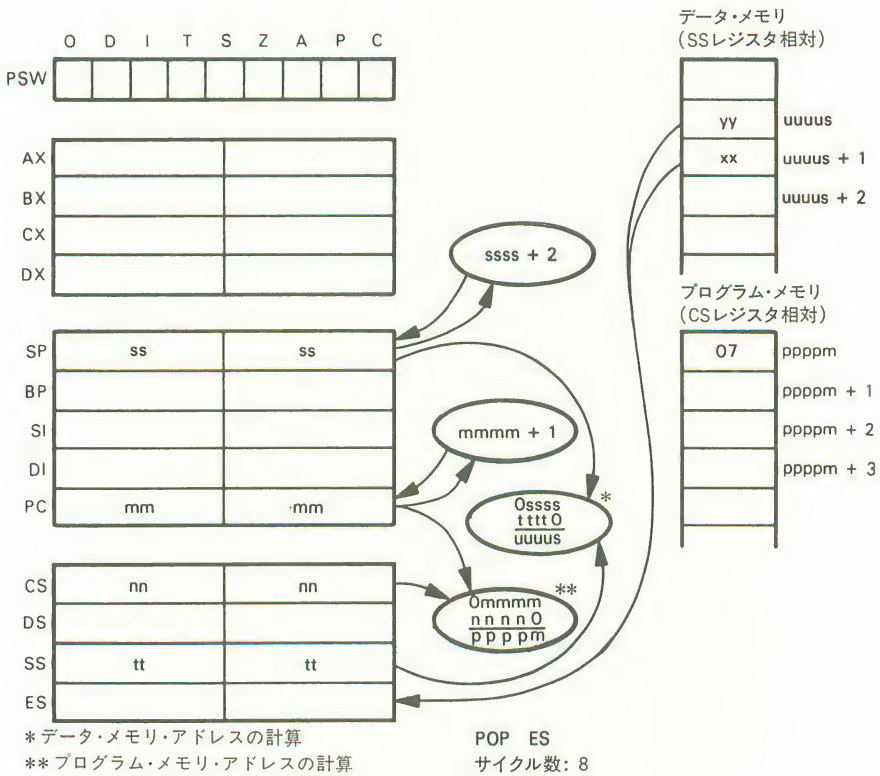
↑
2ビットで、データがポップされる16ビット・セグメント・レジスタを指定

rr=00: ES
10: SS
11: DS

たとえば、

POP ES

の命令は、スタックのトップの2バイトをESレジスタにポップする。rr=01のとき、動作は定義されない。



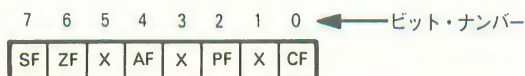
注)

1. この命令は、セグメント・レジスタにのみデータをポップする。8086の他のレジスタにデータをポップするためには、命令 POP reg を参照。
2. POPによって行なわれる機能のより完全な記述については、POP reg を参照。
3. CSに対するPOPは正しくない。CSの変更は、PCへのロードも行なう命令、JMP, CALL, RET, IRET, INTによってのみ行なわれる必要がある。
4. インタラプトは、この命令の終わりではサンプルされない。これに続く命令の終わりにサンプルされる。この制限は、インタラプトを起こさずに、32ビットのポインタ全体の回復を可能にする。これは、SSとSPをロードする際に問題となる。

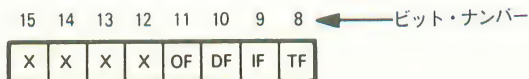
POPF (Pop Flags)

スタックのトップからフラグ・レジスタにリードする。

スタックのトップの2バイトを、フラグ・レジスタにポップする。フラグ・レジスタの下位バイトを以下に示す。



2 番目にポップされるバイトは、フラグ・レジスタの上位バイトにストアされる。このバイトの形式を以下に示す。



命令コードを次に示す。

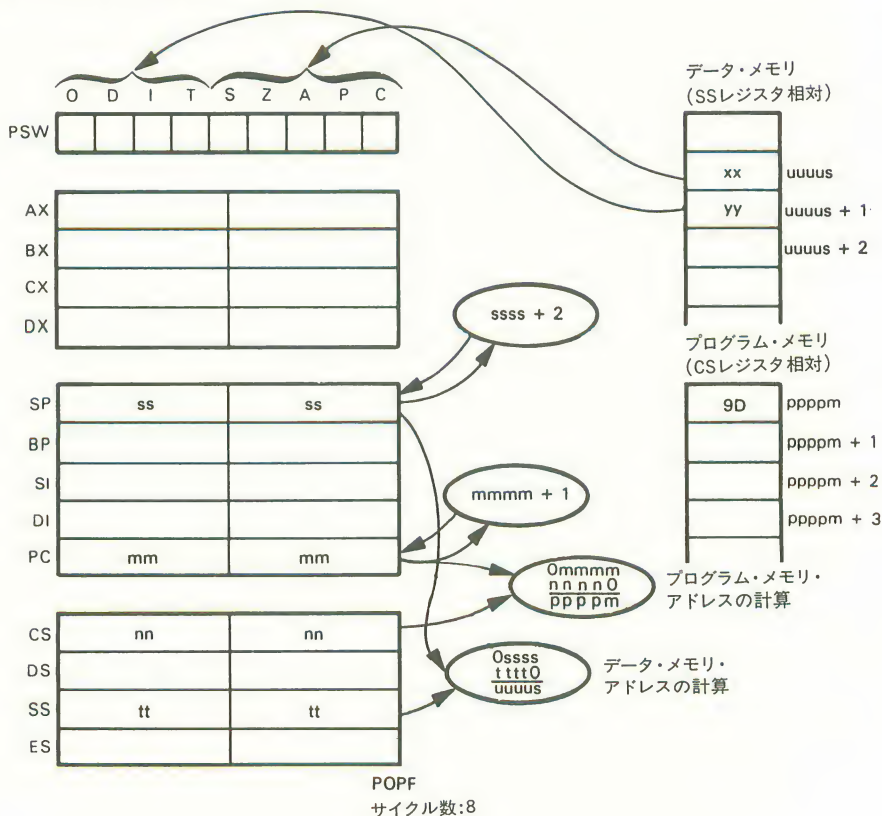
POPF

9D

たとえば、スタックのトップの2 バイトが $4F_{16}$ (最上位) と 32_{16} である場合を考える。

POPF

の実行によって、キャリー、パリティ、ゼロ、インタラプトのフラグは1になり、その他のフラグは0になる。



注)

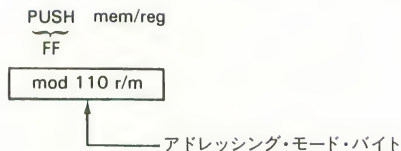
1. すべてのスタックの操作と同じく、スタック・ポインタが初期設定されていることは重要である。さらに、POP命令が実行される前に、フラグの値をストアするためにPUSHF命令が実行されていることが適切である。
2. この命令は、8080の命令POP PSWの機能の一部を果たす*。

PUSH mem/reg (Push)

スタックのトップにライトする。

この命令は、指定されたレジスタあるいはメモリ位置の内容をスタックのトップにプッシュする。これは16ビットのプッシュ操作である。

命令コードを次に示す。

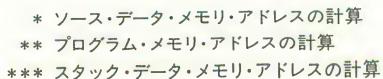


たとえば、DSレジスタが 2800_{16} を含み、BXレジスタが 0400_{16} を含み、SPレジスタが 1000_{16} を含み、SSレジスタが $2F00_{16}$ を含み、メモリ位置 28400_{16} にストアされているワードが $A020_{16}$ を含むとすると、

PUSH [BX]

の命令実行によって、 $A0_{16}$ がメモリ位置 $2FFFF_{16}$ に、 20_{16} がメモリ位置 $2FFFE_{16}$ にストアされる。SPレジスタは $0FFE_{16}$ に調整される。

* 8080の命令POP PSWでは、8ビットのフラグ・レジスタとアキュムレータへのポップを行なう(訳者注)。



サイクル数:メモリ:16 + EA
レジスタ:11

注)

1. この命令は、一般にレジスタをスタックにプッシュするためには用いられない。命令 `PUSH reg` はこの機能を果たし、しかもプログラム・メモリの1バイトのみを占有する。
2. ステータスは影響を受けない。

PUSH reg (Push)

スタックのトップにライトする.

この命令は、指定された16ビット・レジスタの内容を、スタックのトップにプッシュする。

命令コードを次に示す。



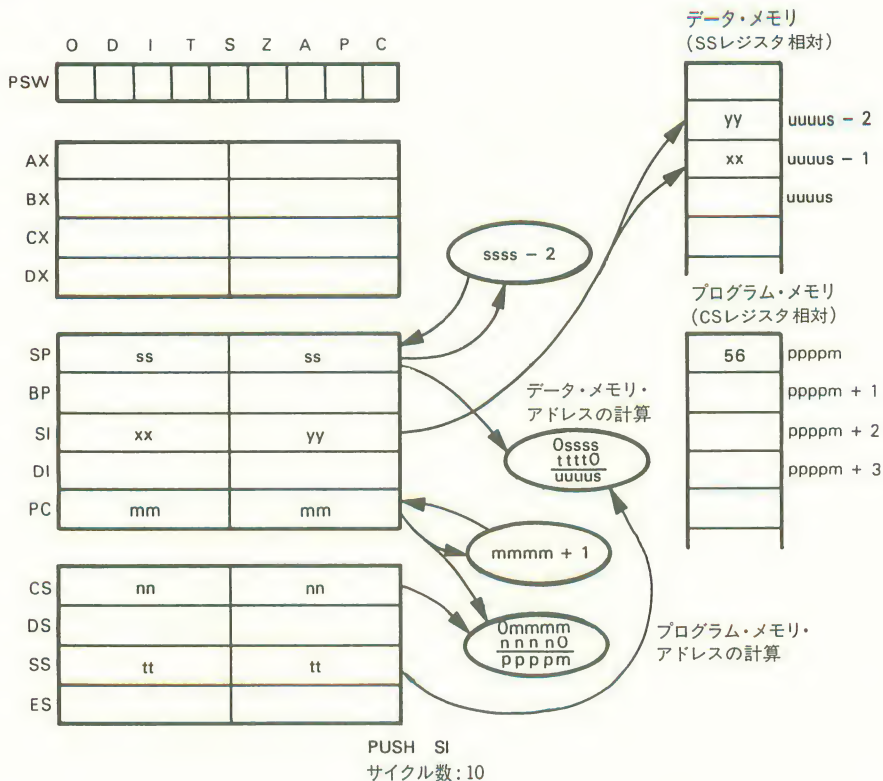
例として、

PUSH SI

の命令を考える。この命令は、SIレジスタの16ビットの内容をスタックにプッシュする。この機能は以下のように行なわれる。

1. スタック・ポインタを1だけ減少。
2. スタック・ポインタとスタック・セグメントで示されるメモリ位置に、指定されたレジスタの上位8ビットをストアする。
3. スタック・ポインタを1だけ減少。
4. スタック・ポインタとスタック・セグメントで示されるメモリ位置に、指定されたレジスタの下位8ビットをストアする。

スタック・ポインタは、一般にスタックのトップと呼ばれるスタックにストアされた最後の要素を示している。



注)

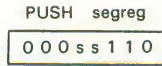
1. この命令は、セグメント・レジスタあるいはフラグ・レジスタのプッシュには使用できない。セグメント・レジスタのプッシュには、命令 PUSH segreg を参照。フラグ・レジスタのプッシュには、PUSHF 命令を参照。
2. この命令は、スタック・ポインタが初期設定された後での使用が最も有効である。事実、この命令は、スタック・ポインタの初期設定後のみ用いられる必要がある。
3. スタックからデータを得るには、POP 命令を用いる。
4. この命令は、8080の命令 PUSH reg と同じ機能を果たす。

PUSH segreg (Push)

スタックのトップにライトする。

この命令は、指定された16ビットのセグメント・レジスタの内容を、スタックのトップにプッシュする。

命令コードを次に示す。



2ビットで、プッシュするセグメント・レジスタを指定。

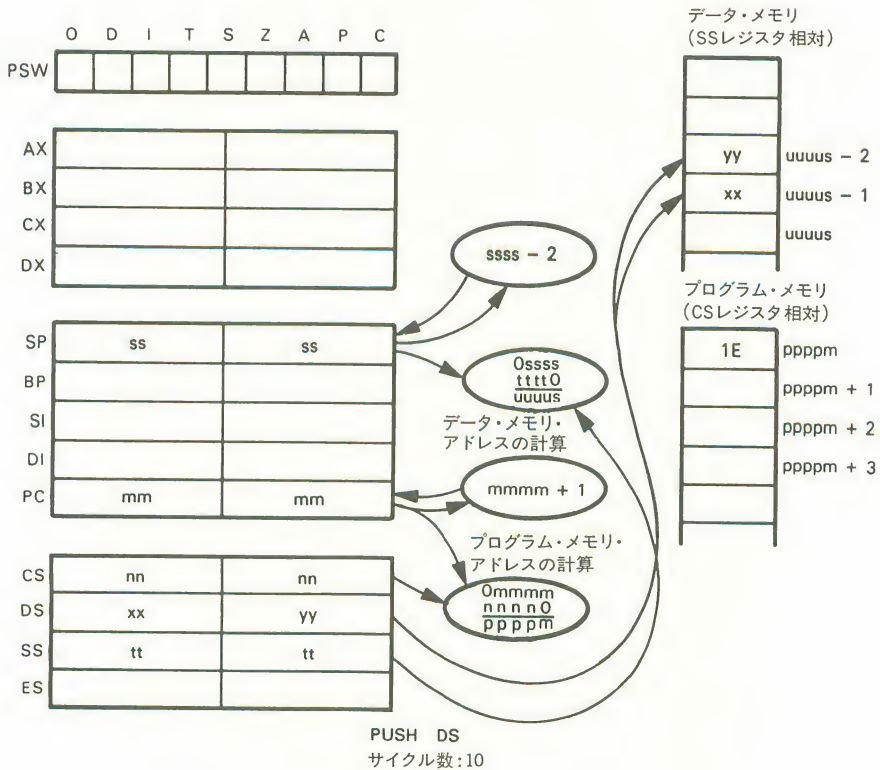
ss = 00:ES
 01:CS
 10:SS
 11:DS

たとえば、次の命令を考える。

PUSH DS

この命令は、DSレジスタの16ビットの内容をスタックにプッシュする。

ss=01のとき、操作は正しくない。



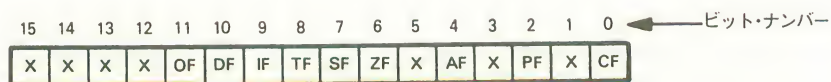
注)

1. この命令は、セグメント・レジスタの内容をスタックにプッシュするためにだけ用いることができる。他のレジスタの内容をプッシュするためには、PUSH regとPUSHFの命令を参照。
3. 適切な結果を保証するためには、スタック・ポインタが初期設定されていなければならないことに注意。

PUSHF (Push Flags)

フラグ・レジスタの内容をスタックのトップにライトする。

この命令は、フラグ・レジスタの内容をスタックのトップにプッシュする。フラグ・レジスタの形式を以下に示す。



ここでXは不定の値を表わす。

最初にビット15・8がスタックに、続いてビット7・0がストアされる。

命令コードを次に示す。

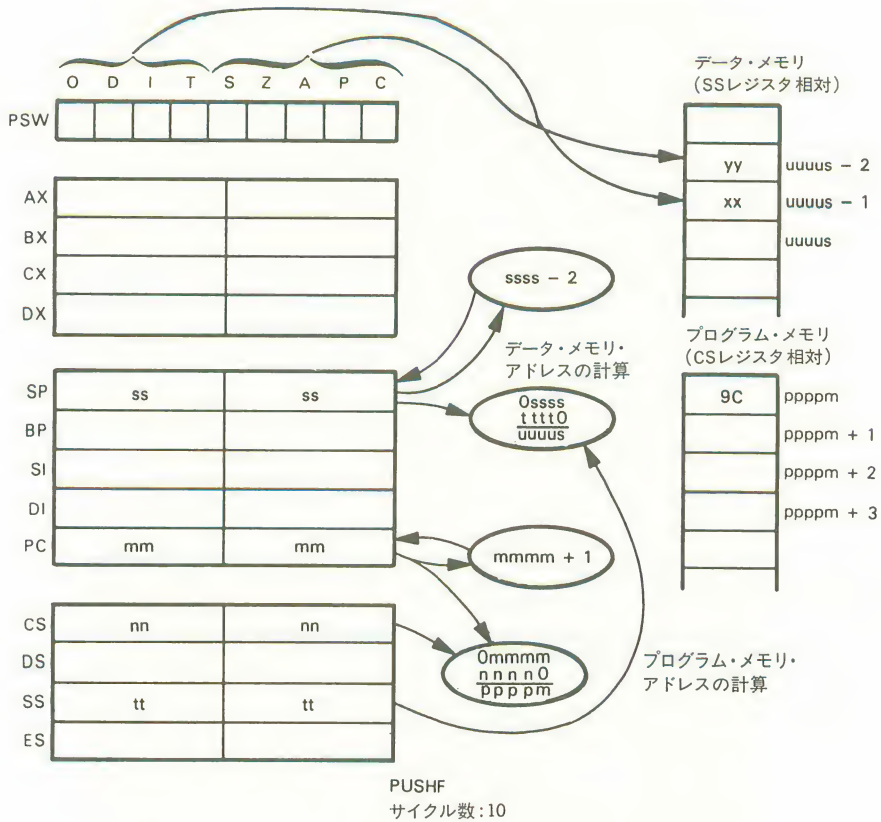
PUSHF
9C

例として、IF、SF、ZFのフラグが1で、一方OF、DF、TF、AF、PF、CFのフラグが0とすると、

PUSHF

の実行によって、以下の操作が行なわれる。

1. スタック・ポインタを1だけ減少。
2. スタック・ポインタとスタック・セグメント・レジスタで示されるメモリ位置に、バイトXXXX0010をストアする(Xは不定の値を表わす)。
3. スタック・ポインタを1だけ減少。
4. スタック・ポインタとスタック・セグメント・レジスタで示されるメモリ位置に、バイト11X0X0X0をストアする。8086に対して、これは1個の16ビット転送を行なっている。



注)

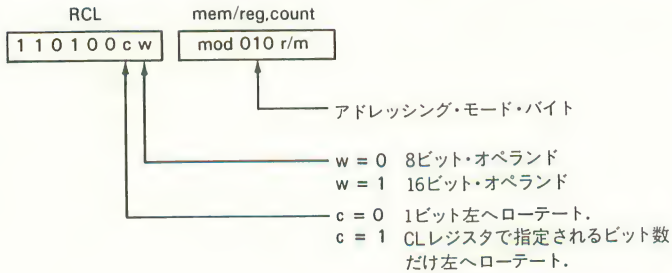
- すべてのスタック命令と同じく、この命令はスタック・ポインタの初期設定が行なわれた後で、最も良くその機能を果たすことに注意。
- この命令は、8080の命令 `PUSH PSW` と同じ機能を果たさない。
`PUSH PSW` 命令は、8080のフラグと同様にアキュムレータの内容をプッシュする。
`PUSH PSW` をエミュレートするには、`LAHF` 命令を参照。

RCL mem/reg,count (Rotate through Carry Left)

キャリーと共にレジスタあるいはメモリの内容を左へローテートする。

指定されたビット数だけ、指定されたレジスタあるいはメモリの内容を、キャリーと共に左へローテートする。変数countで表わされるローテートのビット数は、1あるいはCL

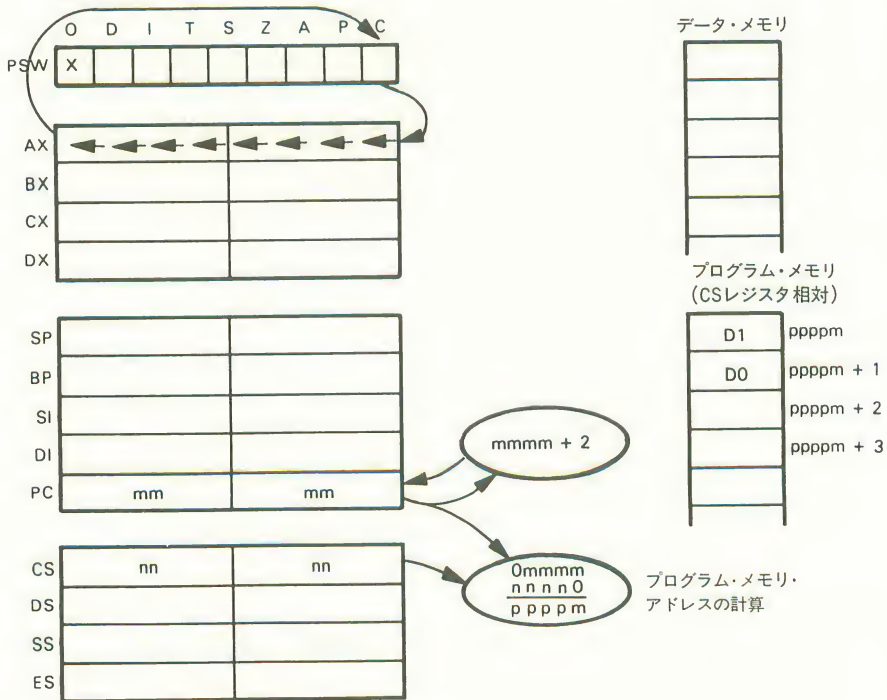
レジスタに含まれる数である。8あるいは16ビットのオペランドが指定できる。
命令コードを次に示す。



A Xレジスタが $FB00_{16}$ を含み、キャリー・フラグが0であるとする。

RCL AX, 1

の実行後、キャリー・フラグは1になり、A Xレジスタは $F600_{16}$ になる。



RCL AX, 1

サイクル数: レジスタ (1ビットのローテート): 2

レジスタ (Nビットのローテート): $8 + (4 * N)$

メモリ (1ビットのローテート): $15 + EA$

メモリ (Nビットのローテート): $20 + EA + (4 * N)$

注)

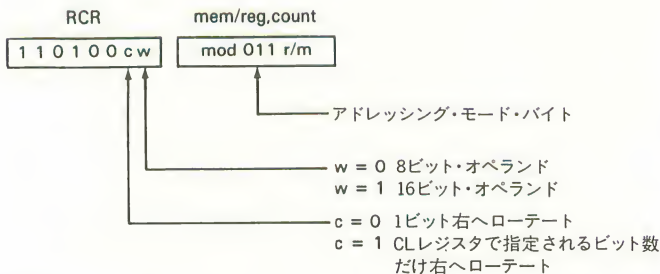
1. この命令は、8080の命令 R A L と同じ機能を果たす。しかしこの命令は、複数ビットのローテートが可能、16ビット数値のローテートが可能、そして任意のレジスタあるいはメモリ位置のローテートができるという点で、非常に大きい融通性がある。
2. 8あるいは16ビットのローテートのどちらが実行されるかは、直観的に明らかではないことに注意。これが決定される方法は、用いるアセンブラに依存している。この興味ある問題の解説は、この章の終わりを参照のこと。

RCR mem/reg,count (Rotate through Carry Right)

キャリーと共にレジスタあるいはメモリの内容を右へローテートする。

指定されたビット数だけ、指定されたレジスタあるいはメモリの内容を、キャリーと共に右へローテートする。変数countで表わされるローテートのビット数は、1あるいはCLレジスタに含まれる数である。8あるいは16ビットのオペランドが指定できる。

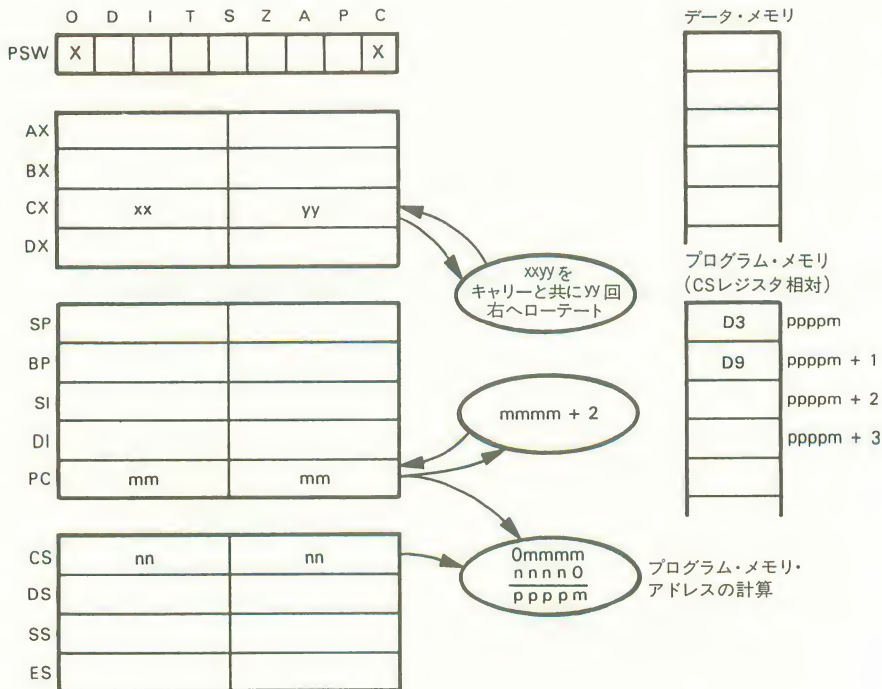
命令コードを次に示す。



CXレジスタがF709₁₆を含み、キャリー・フラグが1であるとする。

RCR CX,CL

の実行後、CXレジスタは09FB₁₆になり、キャリー・フラグは1になる。



RCR CX,CL

サイクル数: レジスタ (Nビットのローテート) : $8 + (4 * N)$
 レジスタ (1ビットのローテート) : 2
 メモリ (Nビットのローテート) : $20 + EA + (4 * N)$
 メモリ (1ビットのローテート) : $15 + EA$

注)

1. この命令は、8080の命令 RAR と同じ機能を果たす。しかしこの命令は、複数ビットのローテートが可能、16ビット数値のローテートが可能、そして任意のレジスタあるいはメモリ位置のローテートができるという点で、非常に大きい融通性がある。
2. この命令を考えると、8あるいは16ビットのローテートの間の差違は明白ではない。この問題の解説は、この章の終わりを参照。

REP/REPE/REPNE/REPNZ/REPZ (Repeat)

後続のストリング命令を繰り返す。

CXレジスタが減少して0になるまで、後続のストリング命令を繰り返す。すべてのストリング命令は、CXが0になるまで実行を続ける。ただし、SCAS と CMPS の命令は例外で、このときは、ZFフラグの値がこの命令の最下位ビット、zビットに等しくなくなれば、実行を中止する。

命令コードを次に示す。

REP/REPE/REPNE

1 1 1 1 0 0 1 z

以下
↑
以下のストリング・プリミティブのとき、Zはドント・ケア・ビット

MOVS

LODS

STOS

以下のストリング・プリミティブにおいて、

CMPS

SCAS

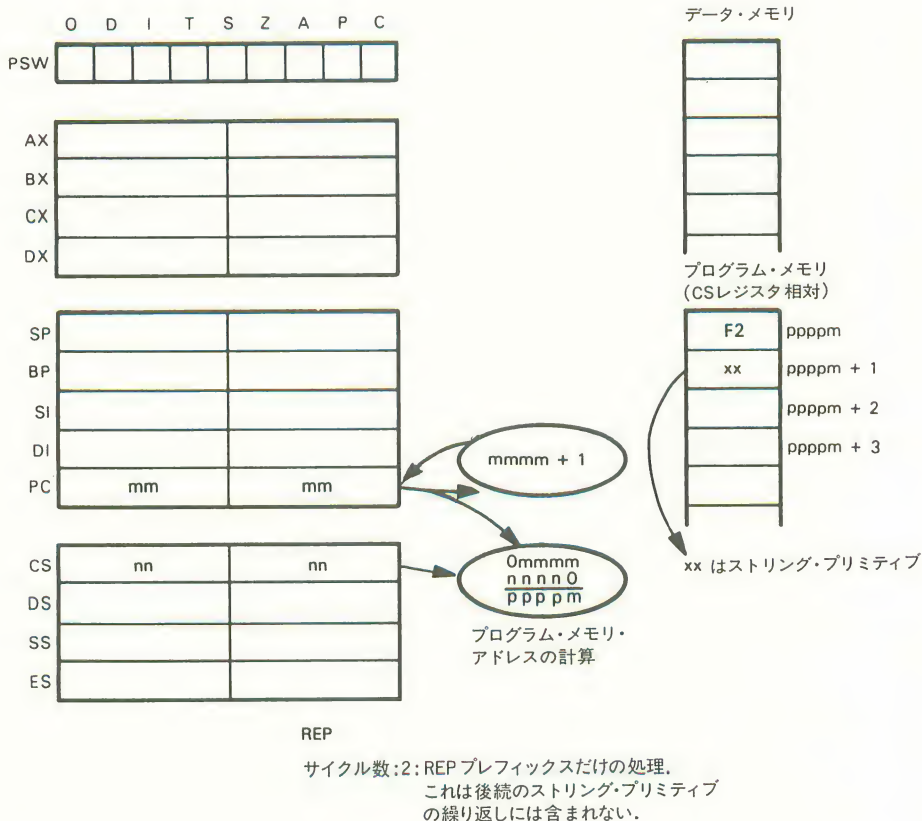
z = 0 ZF = 1ならばCMPSあるいはSCASの命令の実行を終了する。

z = 1 ZF = 0ならばCMPSあるいはSCASの命令の実行を終了する。

以下の一連の命令において、

```
MOV    SI,IOBUF
LES    DI,ADDR
MOV    CX,COUNT
REP
MOVB
```

REP MOVB の命令で、COUNTで表わされるバイト数がIOBUFからADDRに移動される。



注)

1. REPE と REPZ の命令コードは $F3_{16}$ で、REPNE と REPNZ の命令コードは $F2_{16}$ になる。
2. REP は命令プレフィックスと呼ばれる。他のプレフィックスには、LOCK と SEG がある。REP を LOCK あるいは SEG のプレフィックスと共に用いるときは、注意が必要となる。この注意についての詳細は、次章を参照のこと。

RET (Return)

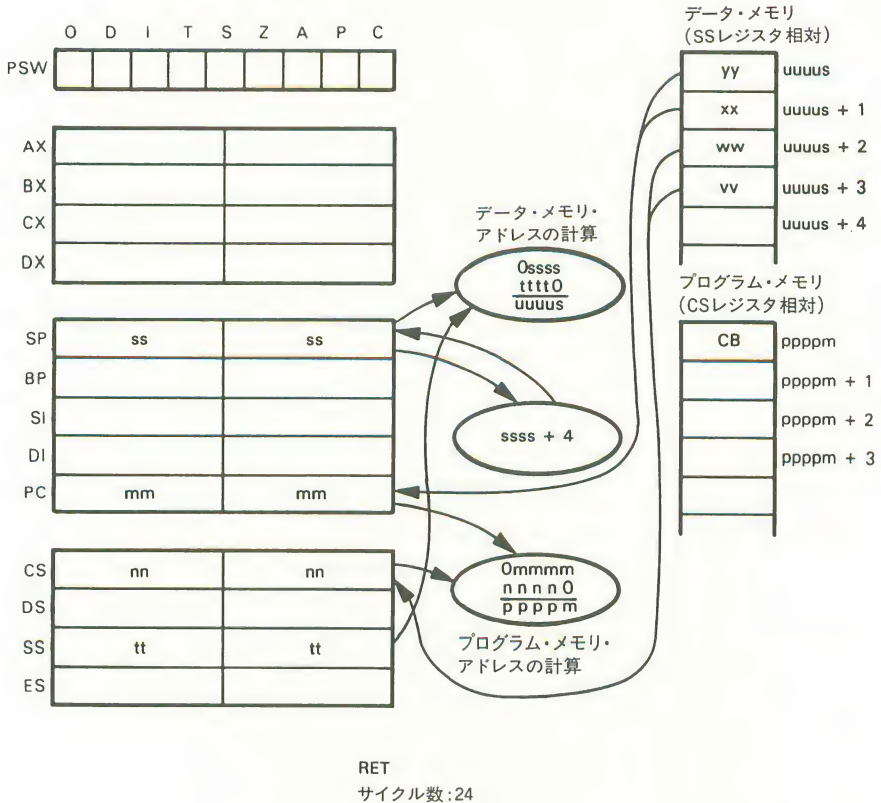
サブルーチン（セグメント間）から復帰する。

スタックのトップの2バイトを、プログラム・カウンタにポップする。この2バイトは、実行されるべき次の命令のオフセット・アドレスを与える。スタックの次の2バイトをC

Sレジスタにポップする。この2バイトは、実行されるべき次の命令のコード・セグメント・アドレスを与える。以前のプログラム・カウンタとコード・セグメント・レジスタの内容は失われる。

命令コードを次に示す。

RET
——
CB



注)

1. すべてのサブルーチンには、少なくとも1個の RET 命令が必要である。この命令は、サブルーチン内で実行される最後の命令であり、制御を呼び出し元プログラムに戻す。
2. この RET 命令は、2つのセグメント間CALL、セグメント間ダイレクトとセグメント間インダイレクトに対応している。
3. ステータスは影響を受けない。

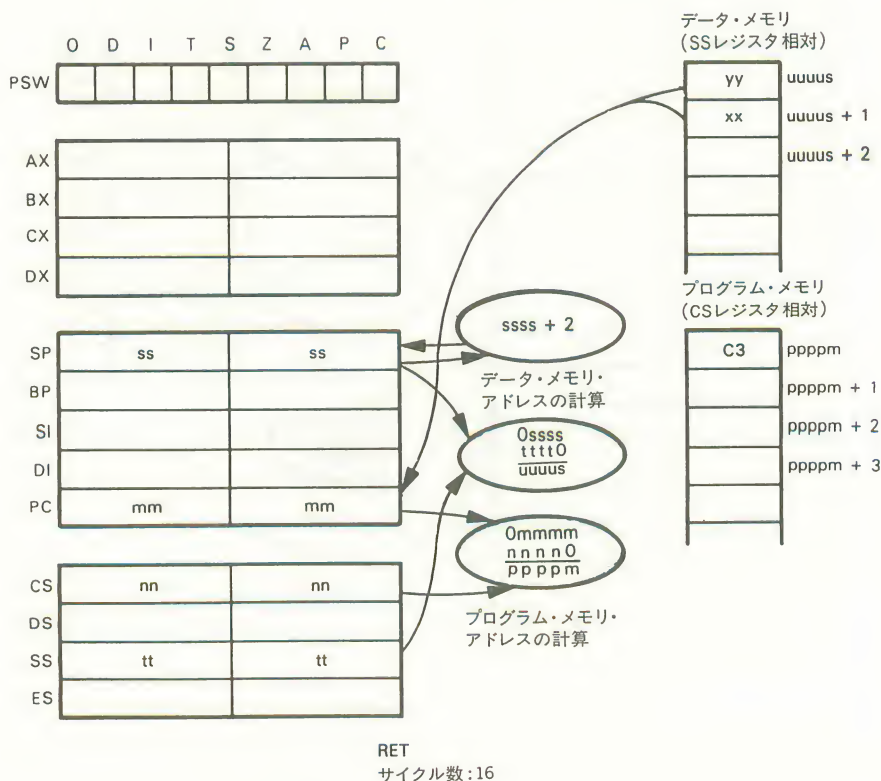
RET (Return)

サブルーチン（セグメント内）から復帰する。

スタックのトップの2バイトの内容をプログラム・カウンタに移動する。すなわち、スタックをプログラム・カウンタにポップする。このバイトは、実行されるべき次の命令のオフセット・アドレスを与える。以前のプログラム・カウンタの内容は失われる。

命令コードを次に示す。

RET
C3



注)

1. この命令は、8080の命令 RET と同じ機能を果たす。
2. すべてのサブルーチンには、少なくとも1個の RET 命令が必要である。これは、サブルーチン内で実行される最後の命令で、実行を呼び出し元プログラムに戻す。呼び

出し元へ戻るために他の方法も可能であるが、一般に簡単な RET 命令と比較して能率が悪く不明瞭となる。

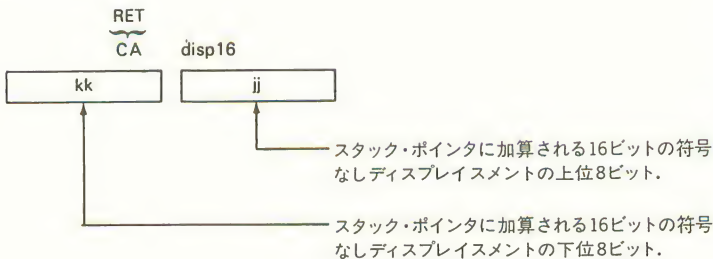
3. 8086には、3つの異なる種類の RET がある。これら RET は、CALL 命令に対してある対応を持っている。この RET は、CALL disp と CALL mem/reg のセグメント内インダイレクトに対応している。
4. ステータスは影響を受けない。

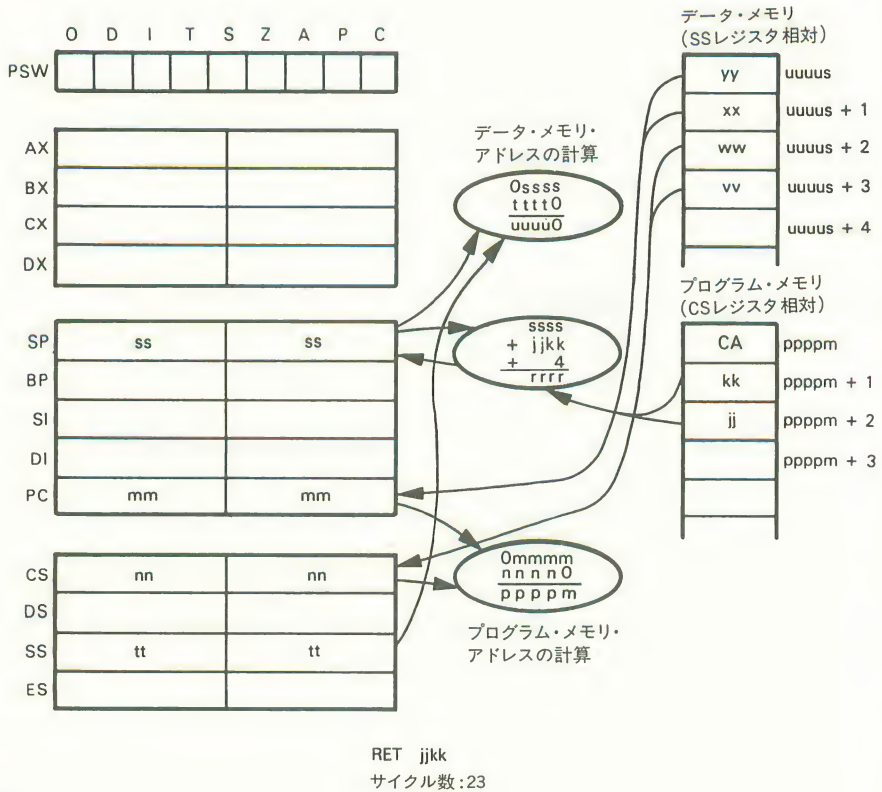
RET disp16 (Return)

サブルーチンから復帰し、スタック・ポインタに加算を行なう（セグメント間）。

スタックのトップの2バイトをプログラム・カウンタにポップする。この2バイトは、実行されるべき次の命令のオフセット・アドレスを与える。次の2バイトをCSレジスタにポップする。この2バイトは、実行されるべき次の命令のコード・セグメント・アドレスを与える。以前のプログラム・カウンタとコード・セグメント・レジスタの内容は失なわれる。後続のプログラム・メモリの2バイトのデータをスタック・ポインタに加算する。これは、このRETに対応するCALLに先立って、スタックに置かれたパラメータを受け渡したスタック・ポインタを調整する意味を持つ。

命令コードを次に示す。





注)

1. ステータスは影響を受けない。
2. すべてのサブルーチンには、少なくとも1つのRET命令が必要である。この命令は、サブルーチン内で実行される最後の命令で、呼び出し元プログラムにおいて対応するCALLの次の命令から実行を再開する。
3. このRET命令は2つのセグメント間CALL、セグメント間ダイレクトとセグメント間インダイレクトに対応する。

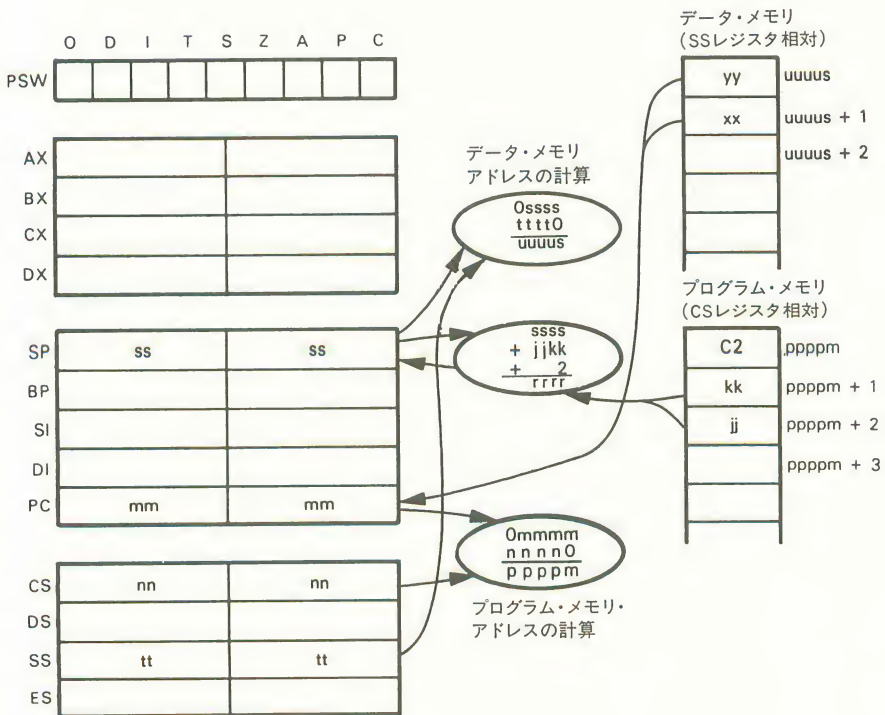
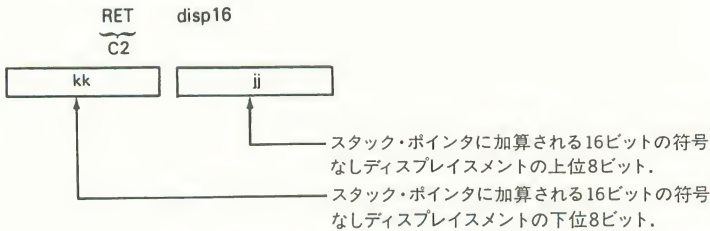
RET disp16 (Return)

サブルーチンから復帰し、スタック・ポインタに加算を行なう (セグメント内)。

スタックからプログラム・カウンタにポップする。移動する2バイトは、実行されるべき次の命令のオフセット・アドレスを与える。以前のプログラム・カウンタの内容は失われる。後続のプログラム・メモリの2バイトのデータをスタック・ポインタに加算する。

これは、このRETに対応するCALLに先立って、スタックに置かれたパラメータを受け渡したスタック・ポインタを調整する意味を持つ。

命令コードを次に示す。



RET jkk
サイクル数:20

注)

1. すべてのサブルーチンには、少なくとも1個のRET命令が必要である。これは、サ

ブルーチン内で実行される最後の命令であり、呼び出し元プログラムに実行を戻す。

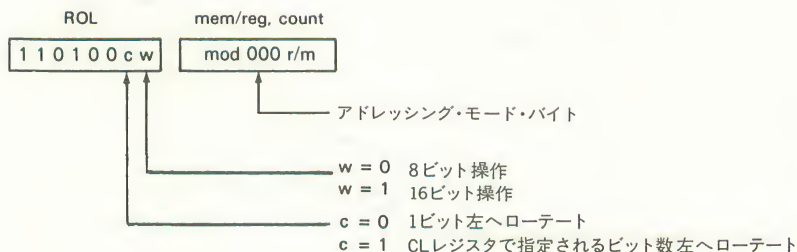
2. 8086には、3つの異なるRETURN命令がある。これらのRETURNにはCALL命令との対応がある。このRETは、CALL disp と CALL mem/regのセグメント内インダイレクトに対応している。
3. ステータスは影響を受けない。

ROL mem/reg, count (Rotate Left)

レジスタあるいはメモリの内容を左へローテートする。

指定されたビット数だけ、指定されたレジスタあるいはメモリ位置の内容を左へローテートする。変数 count で表わされるローテートのビット数は、1あるいはCLレジスタに含まれる数である。

命令コードを次に示す。



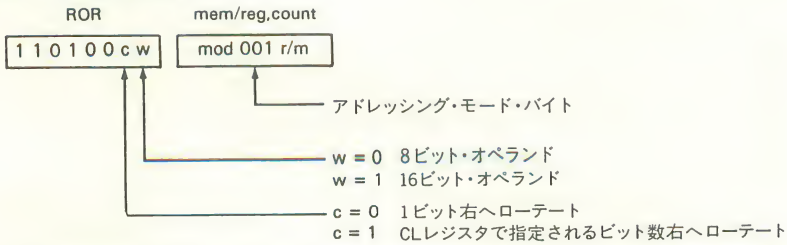
BXレジスタが $AB1F_{16}$ を含み、CLレジスタが 03_{16} を含むとする。

ROL BX, CL

の実行後、BXレジスタは $58FD_{16}$ になり、キャリー・フラグは1になる。

れる数である。

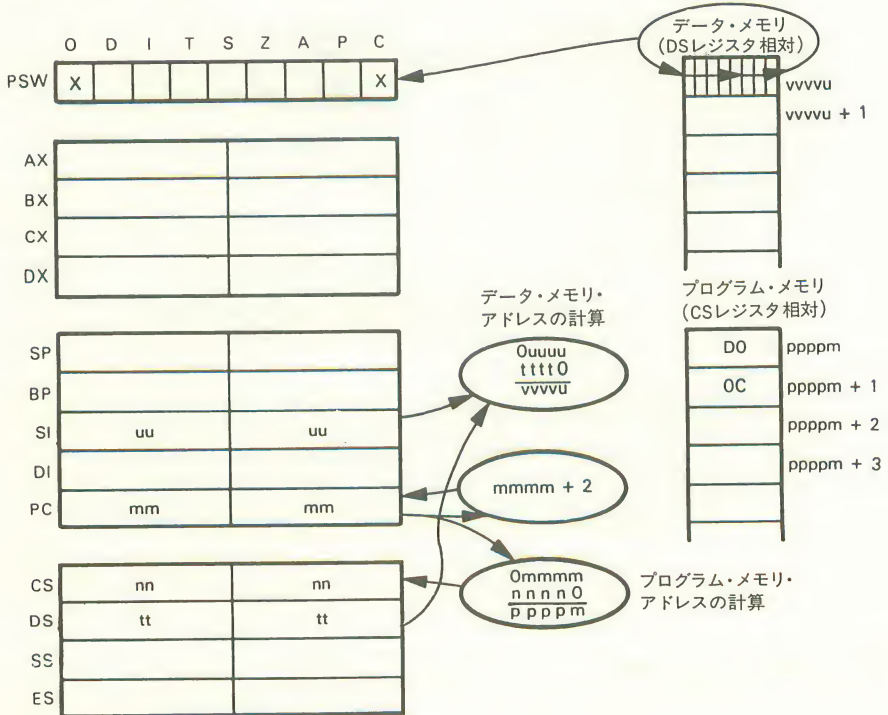
命令コードを次に示す。



DSレジスタがF000₁₆を含み, SIレジスタが06B2₁₆を含み, メモリ位置F06B2₁₆のバイトが04₁₆であるとする。

ROR [SI], 1

の実行後, メモリ位置F06B2₁₆のバイトは02₁₆になり, キャリーとオーバーフローのフラグは0となる。



ROR [SI], 1

サイクル数: メモリ(1ビットのローテート): 15 + EA
メモリ(Nビットのローテート): 20 + EA + (4 * N)
レジスタ(1ビットのローテート): 2
レジスタ(Nビットのローテート): 8 + (4 * N)

注)

1. この命令は、8080の命令 RRC と同じ機能を果たす。しかしこの命令は、複数ビットのローテートが可能、16ビット数値のローテートが可能、そして任意のレジスタあるいはメモリ位置のローテートができるという点で、非常に大きい融通性がある。
2. この命令からは、8あるいは16ビット数値のどちらがローテートされるかを決定できないことに注意。

SAHF (Store AH into 8080 Flags)

AHレジスタを8080のフラグにストアする。

この命令は、AHレジスタの内容をフラグ・レジスタの下位8ビットに移動する。AHレジスタのビットは以下のように用いられる。

ビット7：SFフラグにストア

ビット6：ZFフラグにストア

ビット5：無視

ビット4：AFフラグにストア

ビット3：無視

ビット2：PFフラグにストア

ビット1：無視

ビット0：CFフラグにストア

命令コードを次に示す。

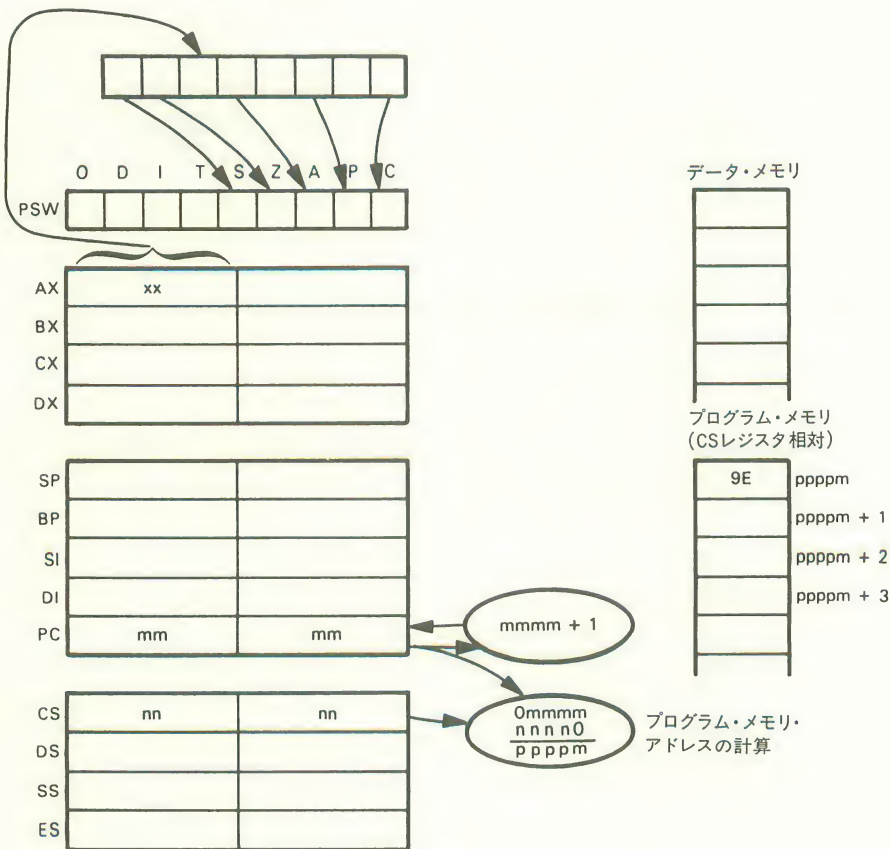
SAHF

9E

たとえば、AHレジスタが $E7_{16}$ を含むとする。

SAHF

の実行によって、SF、ZF、PF、CFのフラグは1になり、一方AFフラグは0になる。



SAHF
サイクル数:4

注)

1. フラグ・レジスタを除いて、レジスタは影響を受けない。OF, DF, IF, TFのフラグは影響を受けない。
2. この命令は、8080の命令 POP PSW をエミュレートするために、POP AX と共に用いられる。

8086コード	8080コード
POP AX	POP PSW
SAHF	

この8086の命令が意味を持つためには、

LAHF
PUSH AX

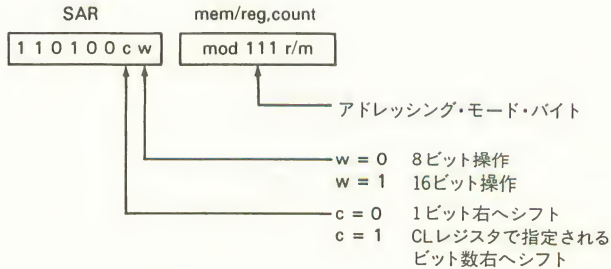
の命令が8080のフラグをセーブするために用いられる必要があることに注意。

SAR mem/reg, count (Shift Arithmetic Right)

レジスタあるいはメモリの内容を右へシフトする。

指定されたビット数だけ、指定されたレジスタあるいはメモリ位置の内容を右へシフトする。変数 count で表わされるシフトのビット数は、1 あるいは CL レジスタに含まれる数である。これは算術的右へのシフトである。

命令コードを次に示す。



CL レジスタが 05_{16} を含み、DI レジスタが $180A_{16}$ を含み、DS レジスタが $F800_{16}$ を含み、メモリ位置 $F980A_{16}$ のワードが 0064_{16} であるとする。

SAR [DI], CL

の実行後、メモリ位置 $F980A_{16}$ のワードは 0003_{16} になる。

	O	D	I	T	S	Z	A	P	C
PSW	X				X	X	?	X	X

AX		
BX		
CX		xx
DX		

SP		
BP		
SI		
DI	uu	uu
PC	mm	mm

CS	nn	nn
DS	tt	tt
SS		
ES		

xxビット右ヘシフト

データ・メモリ・
アドレスの計算

$$\begin{array}{r} 0uuuu \\ tttt0 \\ \hline vvvvu \end{array}$$

$$mmmm + 2$$

$$\begin{array}{r} 0mmmm \\ nnnn0 \\ \hline ppppm \end{array}$$

プログラム・メモリ・
アドレスの計算データ・メモリ
(DSレジスタ相対)

	vvvvu
	vvvvu + 1
	vvvvu + 2

プログラム・メモリ
(CSレジスタ相対)

D3	ppppm
3D	ppppm + 1
	ppppm + 2
	ppppm + 3

SAR [DI],CL

サイクル数: メモリ(Nビットのシフト): $20 + EA + (4 * N)$ メモリ(1ビットのシフト): $15 + EA$ レジスタ(Nビットのシフト): $8 + (4 * N)$

レジスタ(1ビットのシフト): 2

注)

1. 論理的右へのシフトに対して、これは算術的右へのシフトである。その違いを以下に示す。

算術的右へのシフト (SAR)

すべてのビットを右へ一度にシフトする。最上位ビットは同じ状態のままにしておく。これは、最上位ビットの符号拡張の意味を持つ。複数ビットのシフトならば、必要なだけ最上位ビットを符号拡張する。

論理的右へのシフト (SHR)

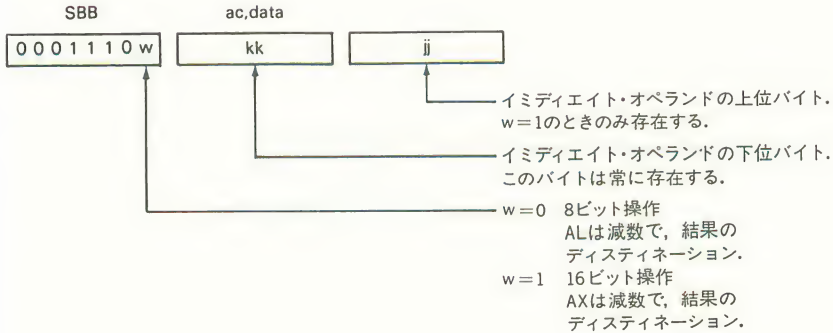
すべてのビットを右へ一度にシフトする。最上位ビットには0をシフトさせる。複数ビットのシフトならば、必要なだけ0をシフトさせることを続ける。

SBB ac, data (Subtract with Borrow)

A XあるいはA Lレジスタからボローと共にイミディエイトを減じる。

A L (8ビット操作) あるいはA X (16ビット操作) のレジスタから、後続のプログラム・メモリ・バイトのイミディエイト・データをボローと共に減じる。減算は2の補数を用いて行なわれる。

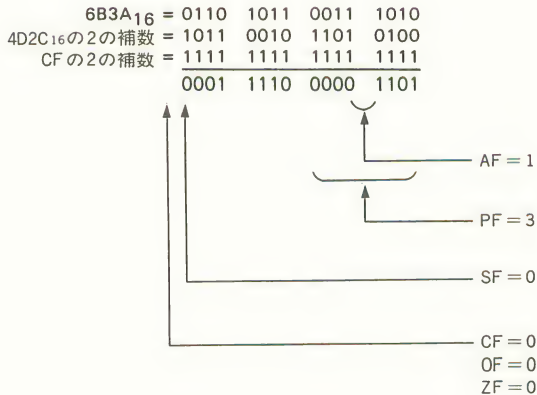
命令コードを次に示す。

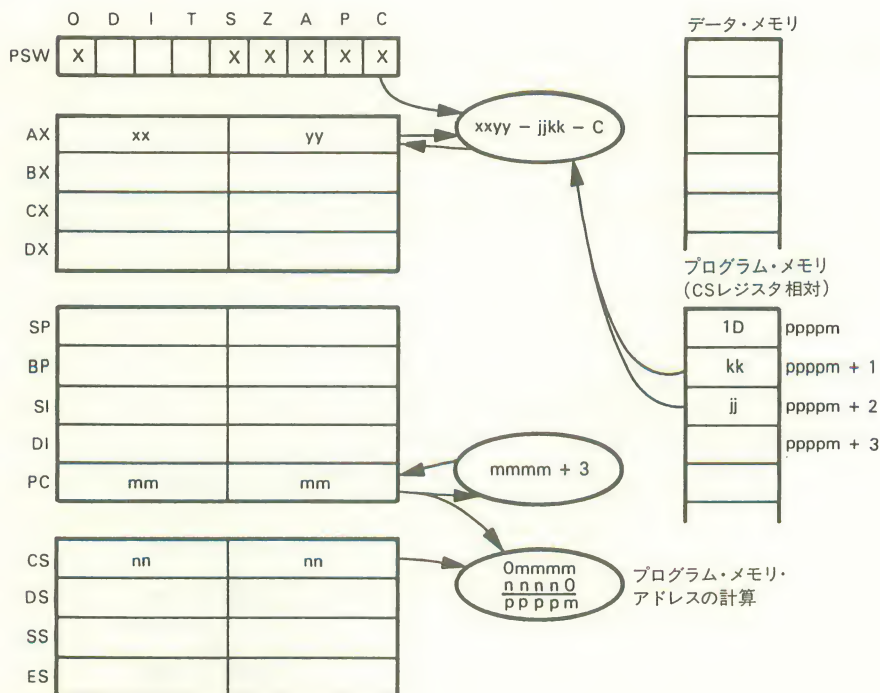


A Xレジスタが $6B3A_{16}$ を含み、キャリー・フラグが1であるとする。

SBB AX, 4D2CH

の実行後、A Xレジスタは $1E0D_{16}$ になる。





SBB AX,jkk

サイクル数:4

注)

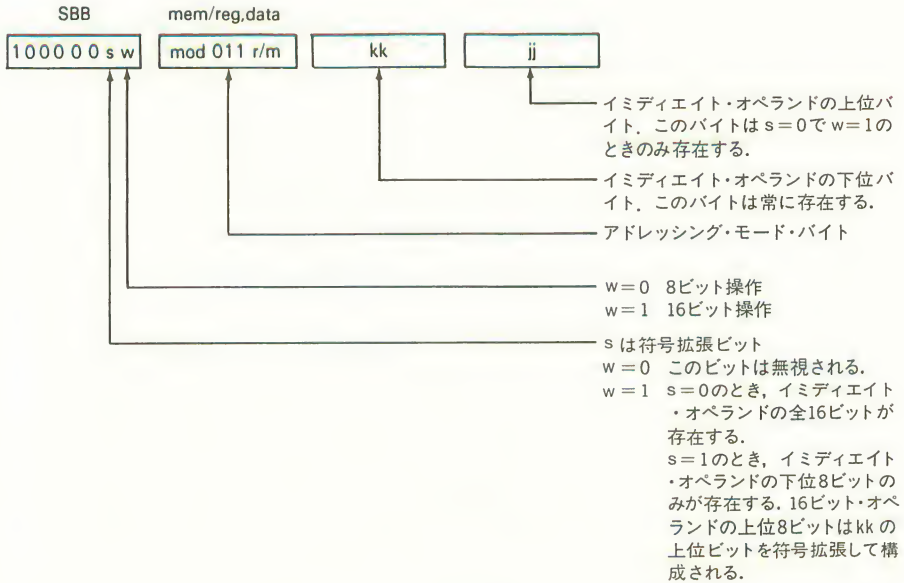
- この命令は、8080の命令 SBI data と同じ機能を果たすが、16ビットの操作も可能である。

SBB mem/reg, data (Subtract with Borrow)

レジスタあるいはメモリからボローと共にイミディエイト・データを減じる。

指定されたレジスタあるいはメモリ位置から、後続のプログラム・メモリ・バイトのイミディエイト・データをボローと共に減じる。8あるいは16ビットの操作が指定できる。減算は、2の補数を用いて行なわれる。

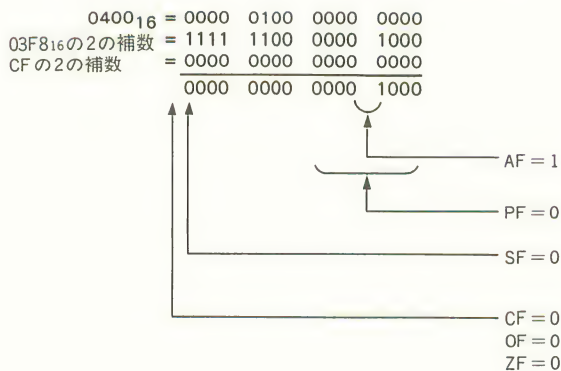
命令コードを次に示す。

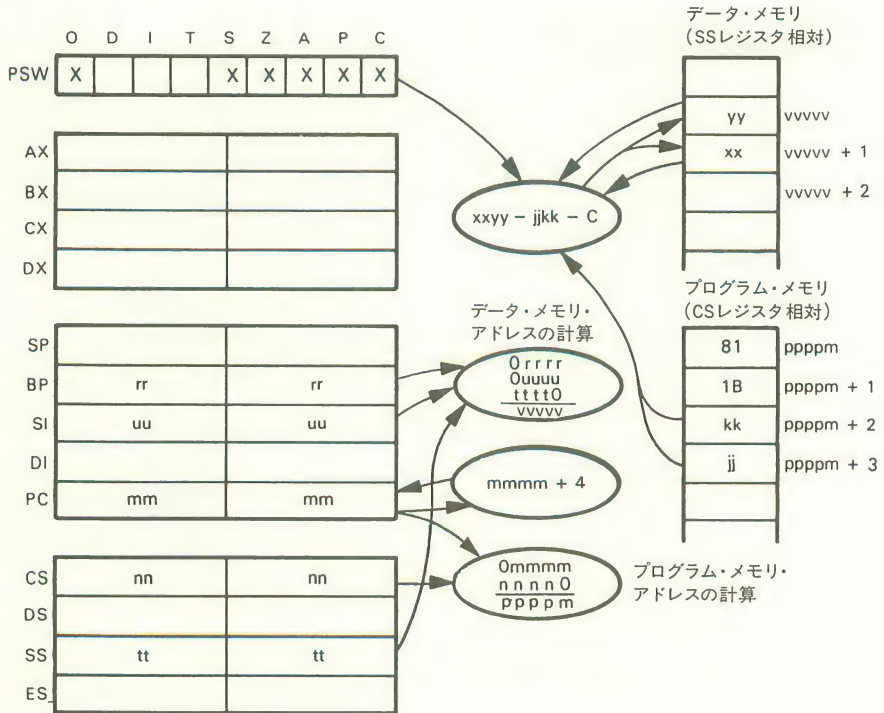


たとえば、キャリー・フラグが0で、SSレジスタが $2F00_{16}$ を含み、BPレジスタが $0F6A_{16}$ を含み、DIレジスタの内容が 0018_{16} で、メモリ位置 $2FF82_{16}$ のワードの内容が 0400_{16} ならば、

SBB [BP+SI], 03F8H

の実行によって、メモリ位置 $2FF82_{16}$ のワードは 0008_{16} に変えられる。





SBB [BP + SI], jkk

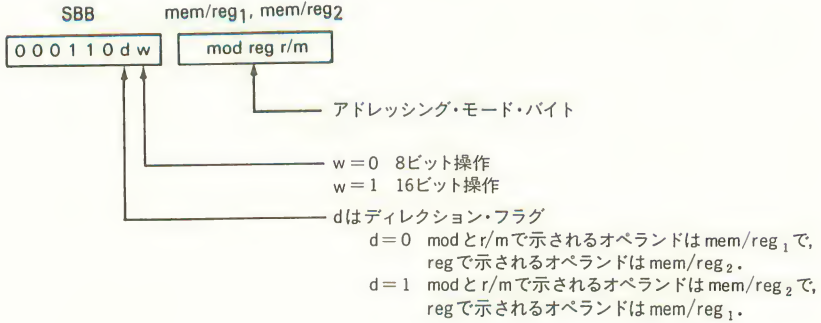
サイクル数: メモリ・オペランド: 17+EA
レジスタ・オペランド: 4

SBB mem/reg₁, mem/reg₂ (Subtract with Borrow)

{ レジスタからレジスタを
 { メモリからレジスタを
 { レジスタからメモリを
 } ボローと共に減じる

mem/reg₁で示されるレジスタあるいはメモリ位置の内容から, mem/reg₂で示されるレジスタあるいはメモリ位置の内容とキャリー・フラグを減じる. 8あるいは16ビットの操作が指定できる. mem/reg₁あるいはmem/reg₂はメモリ・オペランドであるが, オペランドの一方はレジスタ・オペランドでなければならない.

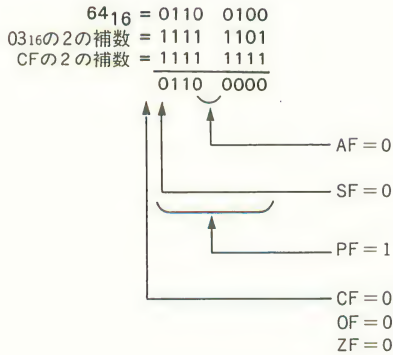
命令コードを次に示す.

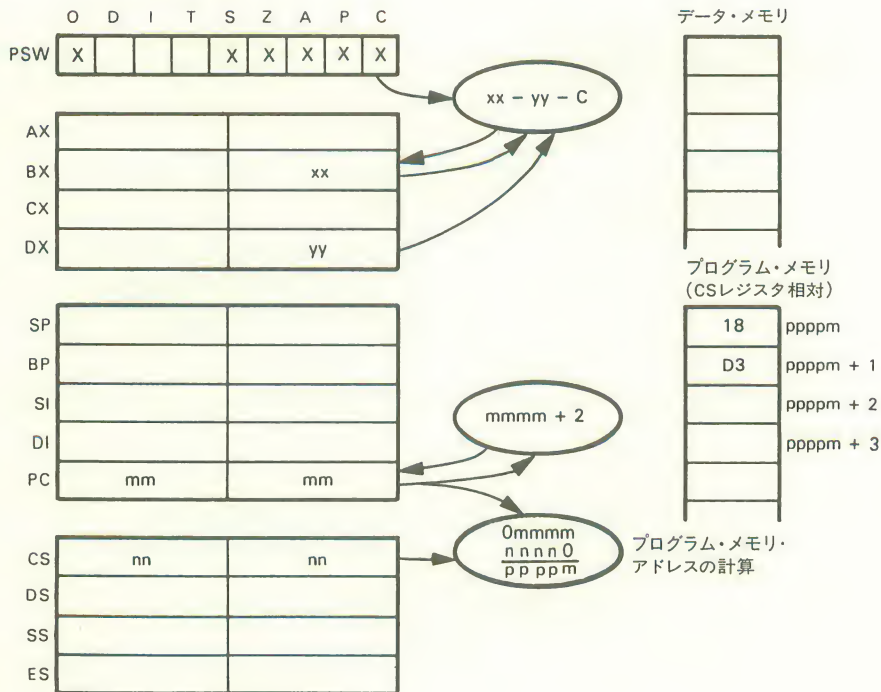


DLレジスタが、03₁₆を含み、BLレジスタが64₁₆を含み、キャリー・フラグが1である場合を考える。

SBB BL,DL

の実行後、BLレジスタは60₁₆になり、フラグは以下のように設定される。





SBB BL, DL

サイクル数: レジスタからレジスタに対して: 3

レジスタからメモリに対して: 16 + EA

メモリからレジスタに対して: 9 + EA

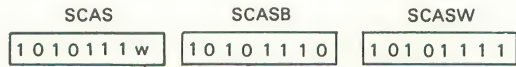
SCAS/SCASB/SCASW

(Scan String)

AL あるいは AX のレジスタとメモリとを比較する。

AL (8ビット操作) あるいは AX (16ビット操作) のレジスタと、DI レジスタで示されるメモリ位置の内容とを比較する。比較は、AL あるいは AX のレジスタから、DI レジスタで示されるメモリ位置の内容を減じ、その結果をフラグに設定することによって行なわれる。メモリあるいは AX レジスタのどちらも変化しない。DI レジスタは、DF フラグの値に応じて増減する。

命令コードを次に示す。



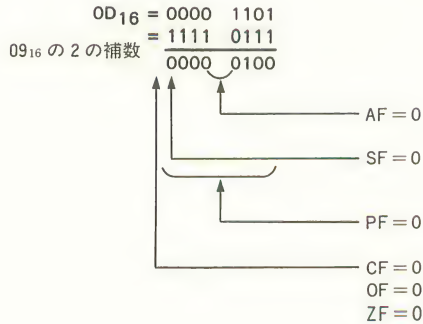
$w = 0$ 8ビットの比較。DF=0のとき、DIレジスタの値は1だけ増加し、DF=1のときは1だけ減少する。

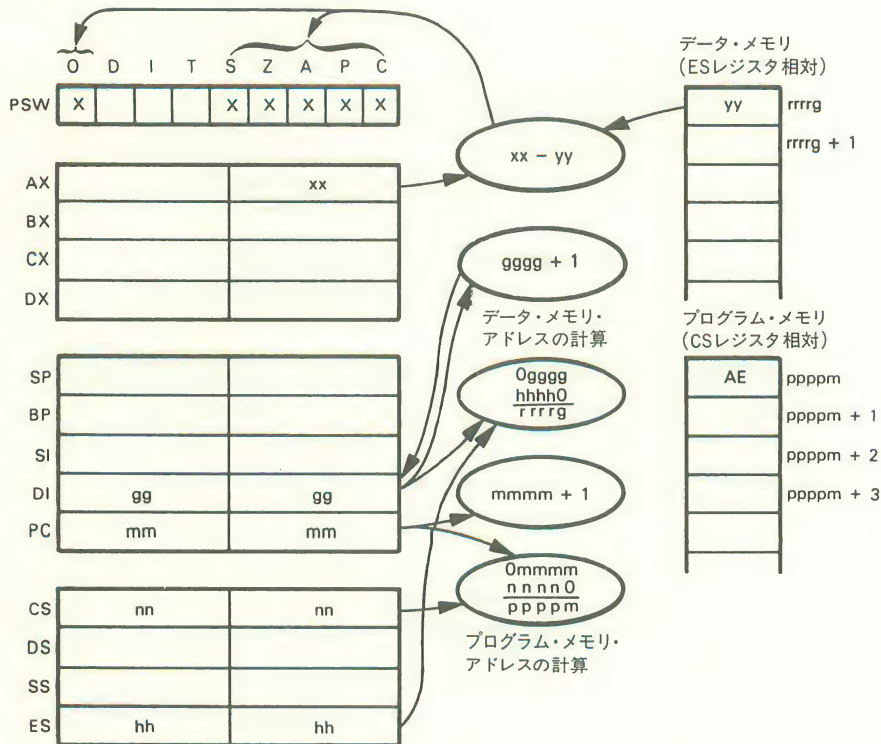
$w = 1$ 16ビットの比較。DF=0のとき、DIレジスタの値は2だけ増加し、DF=1のときは2だけ減少する。

DIレジスタが 0000_{16} を含み、ESIレジスタが 1800_{16} を含み、DFフラグが0で、メモリ位置 18000_{16} の内容が 09_{16} で、ALレジスタの内容が $0D_{16}$ である場合を考える。

SCASB

の命令実行後、DIレジスタは 0001_{16} になり、フラグは以下のように設定される。





SCASB

サイクル数: 15:1度だけ実行したとき

9 + 15 * R: REP プレフィックスによって R 回実行したとき

注)

1. REP プレフィックスや LOCK プレフィックスはこの命令と共に用いられる。REP プレフィックスと LOCK プレフィックスがこの命令と共に用いられた場合は問題となる。この問題の解析は次章に示されている。
2. 汎用形 SCAS のバイトあるいはワードのオペランドの決定は、この章の終わりで論じている。

SEG segreg (Segment)

デフォルト・セグメント・レジスタを変更する。

このプレフィックスが先行した命令のデータ・メモリ・アドレスの計算に、指定されたセグメント・レジスタを用いる。すなわち、データ・メモリ・アドレスの計算に、セグメント・アドレスとして指定されたセグメント・レジスタの内容を用いる。

命令コードを次に示す。

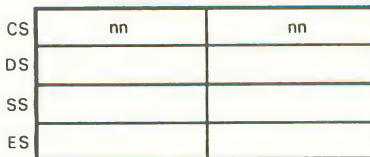
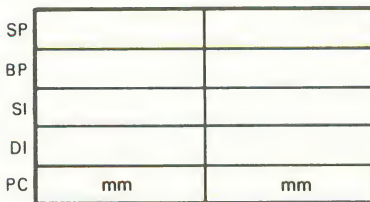
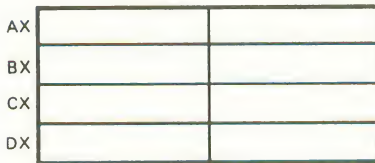
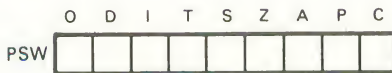


ss = 00:ES
 01:CS
 10:SS
 11:DS

DSレジスタが 1000_{16} を含み、ESレジスタが 2000_{16} を含み、BXレジスタが 0008_{16} を含み、メモリ位置 10008_{16} のワードが $FEFE_{16}$ で、メモリ位置 20008_{16} のワードが $060A_{16}$ である場合を考える。

SEG ES
 MOV AX, [BX]

の実行後、AXレジスタは $060A_{16}$ になる。



mmmm + 1

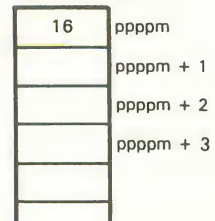
0mmmm
 nnnnn0
 ppppm

プログラム・メモリ・
 アドレスの計算

データ・メモリ



プログラム・メモリ
 (CSレジスタ相対)



SEG ES
 サイクル数:2

注)

1. セグメント変更プレフィックスの導入はアセンブラに依存する。前に示したように、DSをESに変更するために、

```
SEG ES
MOV AX, [BX]
```

が用いられる。

インテルのアセンブラで用いられている他の方法は、

```
MOV AX, ES:[BX]
```

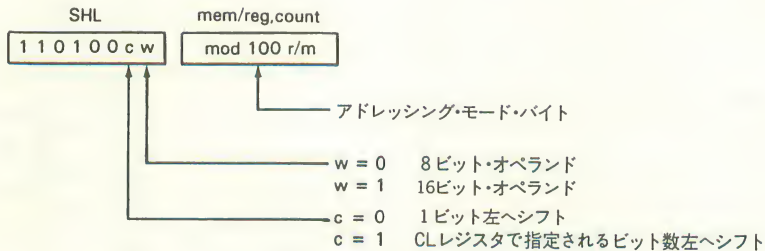
で、セグメント変更を移動命令中に組み込むことを要求している。アセンブラは、MOV命令コード生成の一部として変更プレフィックスを生成する。

SHL *mem/reg, count* (Shift Left)
SAL *mem/reg, count*

レジスタあるいはメモリの内容を左へシフトする。

指定されたビット数だけ、指定されたレジスタあるいはメモリの内容を左へシフトする。変数 *count* で表わされるシフトのビット数は、1 あるいは CL レジスタに含まれる数である。これは論理的な左へのシフトである。

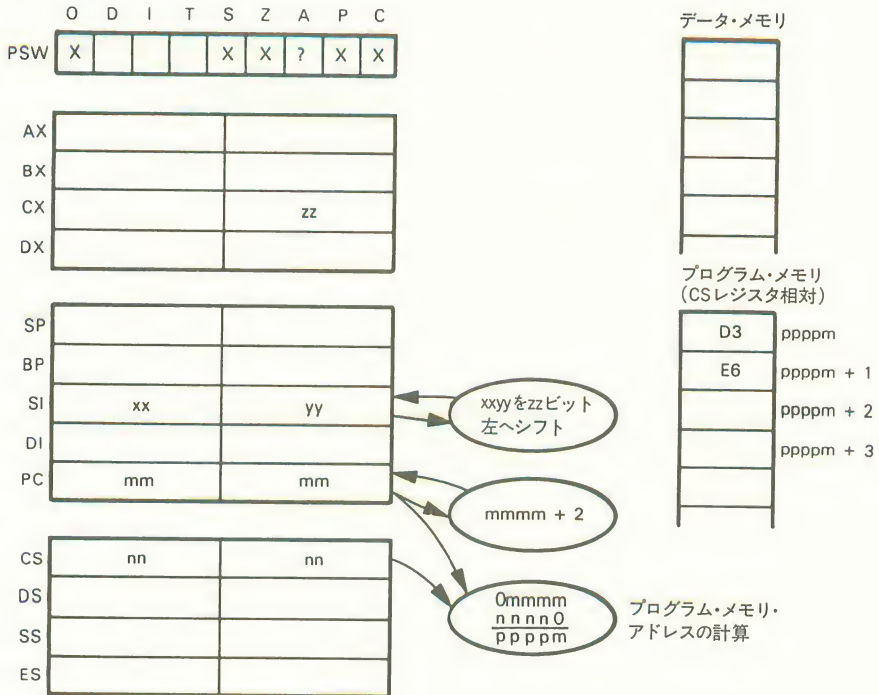
命令コードを次に示す。



CL レジスタが 02_{16} を含み、SI レジスタが $A450_{16}$ を含むとする。

```
SHL SI, CL
```

の実行後、SI レジスタは 9140_{16} になり、キャリー・フラグは 0 になる。



SHL SI,CL

サイクル数: レジスタ(Nビットのシフト): $8 + (4 * N)$

レジスタ(1ビットのシフト): 2

メモリ (Nビットのシフト): $20 + EA + (4 * N)$ メモリ (1ビットのシフト): $15 + EA$

注)

1. この命令は、シフトによる加算で乗算を行なうために用いることができる。MULとIMULの命令は、実行に少なくとも71サイクルを必要とし、乗算を行なうのにシフトを用いるのが興味ある方法となる場合がある。代表的には、このような場合は、メモリの保存よりも処理速度の最適化が大きい要素となるときや、実行されるべき乗算が常に2のべき乗あるいは常に定数のときに生じる。以下に示すいくつかの場合を考える。

```
CALL  MULT$BY$8
-
-
-
MULT$BY$8 MOV  CL,3
SAL      AX,CL
RET
```

MULT\$BY\$8のルーチンはそのルーチンのコードに5バイトを必要とし、CALLのコードには3バイトを必要とする。しかし、乗算を行なうのに、71サイクル（最小）を必要とする代わりに、CALLには19サイクルを要し、ルーチンには32サイクルを要する。

```
CALL  SAL$THREES$TIMES
-
-
-
SAL$THREES$TIMES  SAL
                   SAL
                   SAL
                   RET
```

上記のルーチンはさらに2バイトを必要とするが、このルーチンは14サイクルで実行される。

2のべき乗を乗じるだけのルーチンの選択は確かにSHL命令を引き立たせていることは明らかである。次に15を乗じる場合を考える。

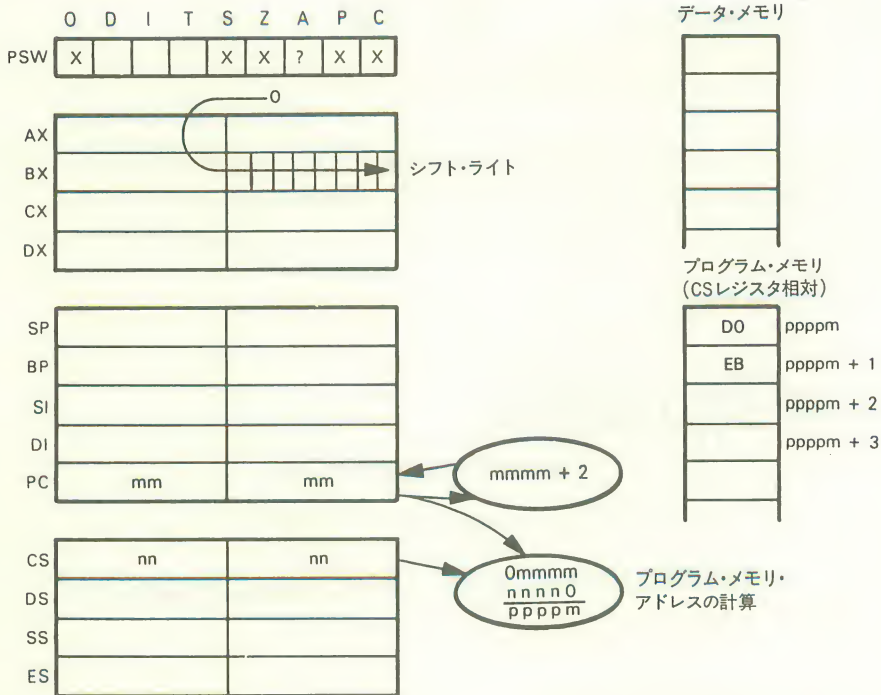
```
CALL  MULT$BY$15
-
-
MULT$BY$15  MOV    CL,4
             MOV    DX,AX
             SAL     AL,CL
             SUB     AX,DX
             RET
```

このルーチンは、9バイトのコードと41サイクル、それにCALLの19サイクルを必要とする。これは、MUL命令を用いるよりもほんのぎりぎりだけ速い。このルーチンは、個々のSAL命令を含むことによってずっと速く動作できる。

```
CALL  MULT$BY$15
-
-
MULT$BY$15  MOV     DX,AX
             SAL
             SAL
             SAL
             SUB     AX,DX
             RET
```

この場合、ルーチンは動作に21サイクルしか要しない。

2. 8あるいは16ビットのどちらのローテーションかは、この命令の記述に用いられている表現方法では、指定されていない。



SHR BL

サイクル数: レジスタ(1ビットのシフト): 2
 レジスタ(Nビットのシフト): $8 + (4 * N)$
 メモリ (1ビットのシフト): $15 + EA$
 メモリ (Nビットのシフト): $20 + EA + (4 * N)$

注)

1. 算術的な右へのシフトに対して、これは論理的な右へのシフトである。その違いを以下に示す。

論理的右へのシフト (SHR)

すべてのビットを右へ一度にシフトする。最上位ビットには0をシフトさせる。複数ビットのシフトならば、必要なだけ0をシフトさせることを続ける。

算術的右へのシフト (SAR)

すべてのビットを右へ一度にシフトする。最上位ビットは同じ状態のままにしておく。これは、最上位ビットの符号拡張の意味を持つ。複数ビットのシフトならば、必要なだけ最上位ビットを符号拡張する。

STC (Set Carry flag)

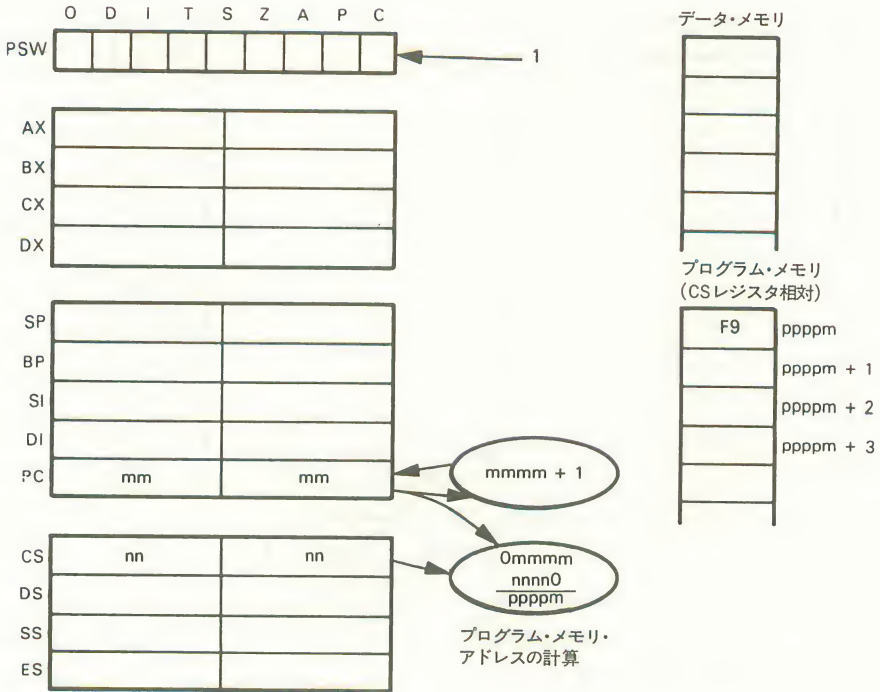
キャリー・フラグをセットする。

この命令は、キャリー・フラグを1にするために用いられる。他のステータスあるいはレジスタの内容は影響を受けない。

命令コードを次に示す。

STC

~~~~~  
F9



STC

サイクル数: 2

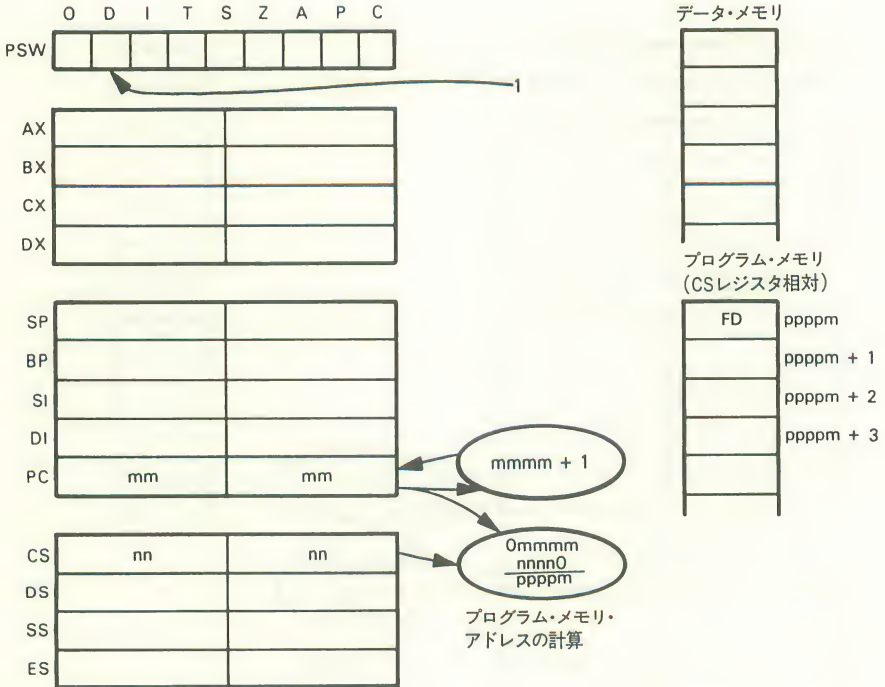
# STD (Set Direction flag)

ディレクション・フラグをセットする。

この命令は、ディレクション・フラグを1にするために用いられる。他のステータスあるいはレジスタの内容は影響を受けない。この命令は、ストリング操作で、用いられているポインタを自動減少にする。

命令コードを次に示す。

STD  
FD



STD  
サイクル数: 2

## STI (Set Interrupt flag)

インタラプト・フラグをセットする。

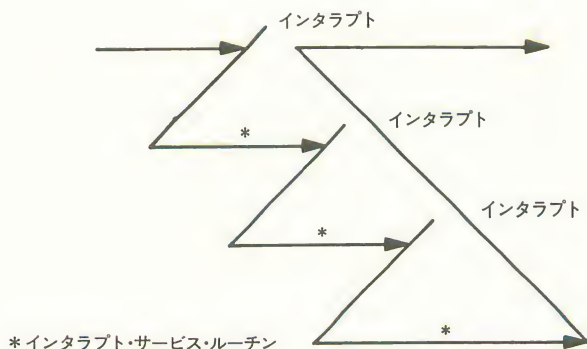
後続の命令実行後に、インタラプト・フラグを1にする。これはインタラプトを可能にする効果を持つ。

1命令待つ理由は以下のとおりである。多くのインタラプト・サービス・ルーチンは次の2つの命令で終了する。

```
STI      ;ENABLE INTERRUPTS
RET      ;RETURN TO INTERRUPTED PROGRAM
```

もしインタラプトが連続的に処理されるならば、インタラプト・サービス・ルーチンの全期間、すべてのインタラプトは無効となる。このことは、複数インタラプトの応用において、どれかインタラプト・サービス・ルーチンが実行を終了するときに、1つあるいは複数のインタラプトが未処理になる重大な可能性の存在を意味している。

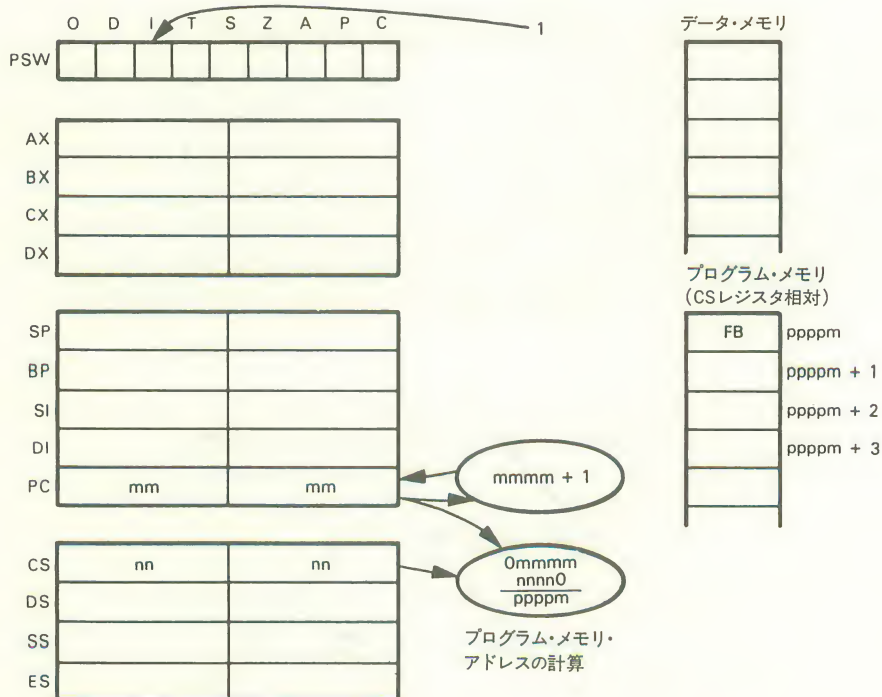
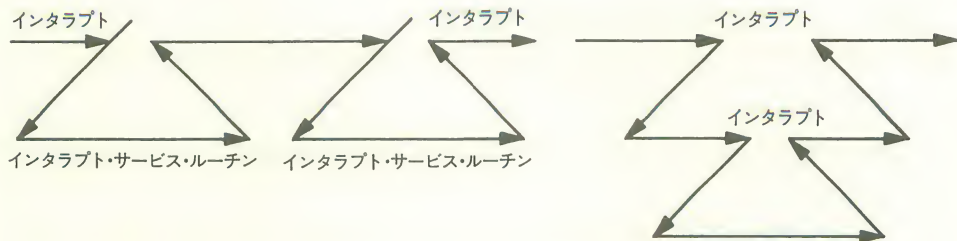
もしSTI命令が実行されるとただちにインタラプトが受け付けられたならば、リターン命令は実行されないかもしれない。このような状況では、次から次へとスタックが行なわれ、不必要にスタック・メモリ領域を消費する。これは以下のように図示できる。



STIに続くさらに1つの命令実行の間インタラプトを禁止することによって、8086CPUはRET命令が続いて実行されることを確実にする。

```
STI      ;ENABLE INTERRUPTS
RET      ;RETURN FROM INTERRUPT
```

インタラプト・サービス・ルーチンを実行している間、インタラプトを無効にしておくことは珍しくなく、インタラプトは連続して処理される。



STI  
サイクル数: 2

注)

1. この命令は、8080の命令 E I と同じ機能を果たす。

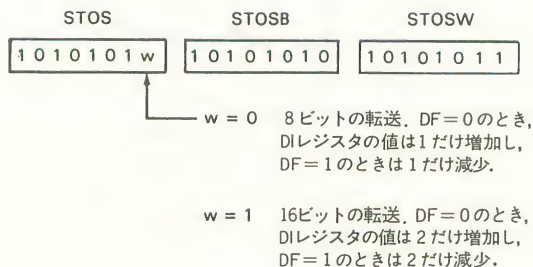


## STOS/STOSB/STOSW (Store String)

ALあるいはAXのレジスタの内容をメモリにストアする。

AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を、DIレジスタで示されるメモリ位置にストアする。DIレジスタは、DFフラグの値に依存して増減する。

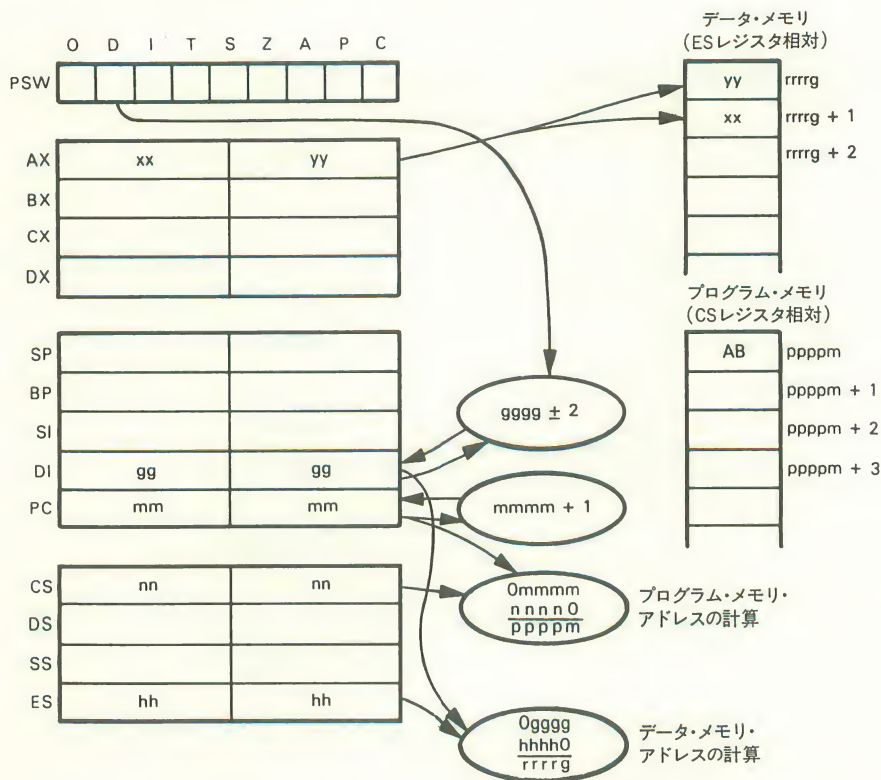
命令コードを次に示す。



たとえば、DFフラグが1で、DIレジスタが $000A_{16}$ を含み、ESレジスタが $2800_{16}$ を含み、AXレジスタが $0604_{16}$ を含むと仮定する。

STOW

の実行後、メモリ位置 $2800A_{16}$ のワードの内容は $0604_{16}$ になり、DIレジスタは $0008_{16}$ になる。

**STOSW**

サイクル数: 11: 1度だけ実行のとき

9 + (10 \* R): REP プレフィックスによって R 回実行したとき

注)

1. ステータスは影響を受けない。
2. この命令のセグメント・アドレスは、常に ES レジスタに含まれる。この命令には、セグメント変更プレフィックスは使用できない。もしセグメント変更プレフィックスがあっても無視される。
3. この命令には、REP プレフィックスや LOCK プレフィックスが先行できる。REP と LOCK プレフィックスをこの命令と共に用いることは問題となる。この潜在的な問題の完全な解説は次章を参照。
4. この命令は、バッファあるいはデータ領域の全体を特定の値に設定するのに非常に有用である。以下の一連の命令を考える。

```

LES      DI, JOB$COSTING$ARRAY
MOV      CX, JOB$COSTING$ARRAY$WORD$LENGTH
MOV      AX, 0000H

```

REP  
STOS WORD

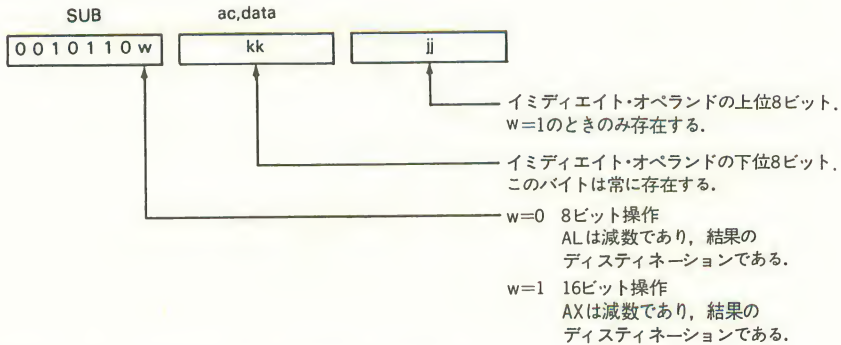
この命令の実行後、JOB\$COSTING\$ARRAYはすべて0を含む。

5. STOSの汎用形に対して、8あるいは16ビットのどちらをストアするかをアセンブラがどのように決めるかについての解説は、この章の最後の節を参照。

## SUB ac,data (Subtract)

ALあるいはAXのレジスタからイミディエイト・データを減じる。

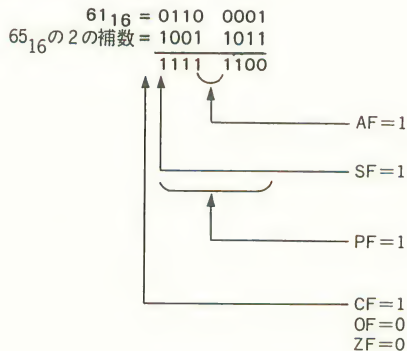
この命令は、AL（8ビット操作）あるいはAX（16ビット操作）のレジスタからイミディエイト・データを減じるために用いられる。減算は2の補数表示を用いて行なわれる。命令コードを次に示す。



たとえば、ALレジスタが $61_{16}$ を含むと仮定する。

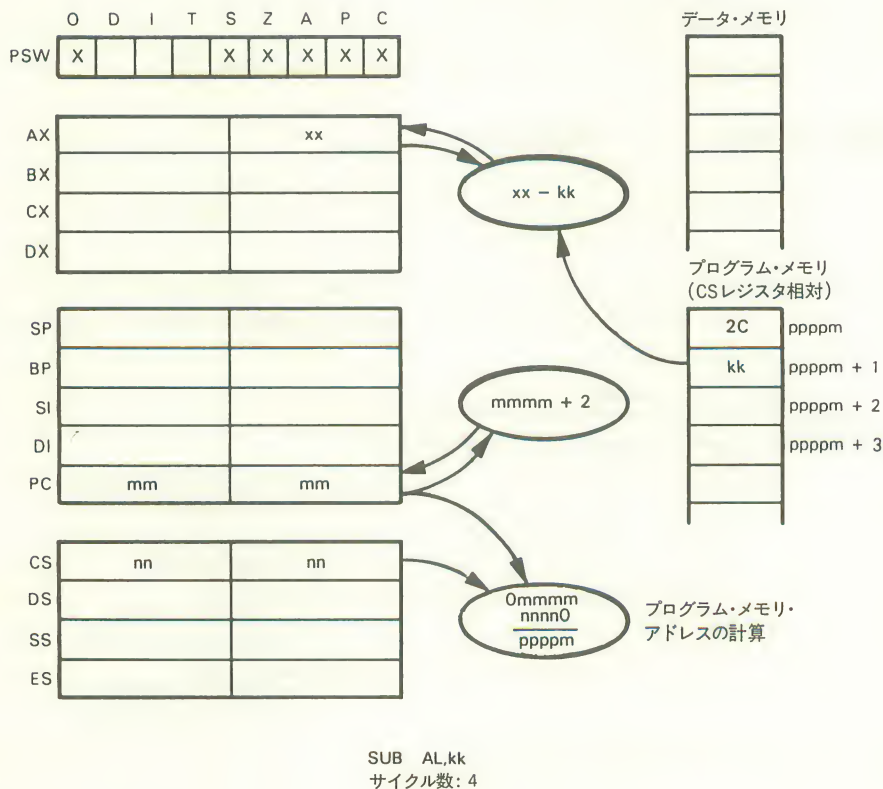
SUB AL,65H

の実行後、ALレジスタの内容は $FC_{16}$ になる。



結果のキャリーは補数がとられることに注意。

FC<sub>16</sub>は-4の2の補数表示であり、これは実際 61<sub>16</sub> から 65<sub>13</sub> を引いたときの結果になっていることに注意。



注)

1. この命令は、8080の命令 SUI data と同じ機能を果たす。さらに、16ビットの操作も可能である。

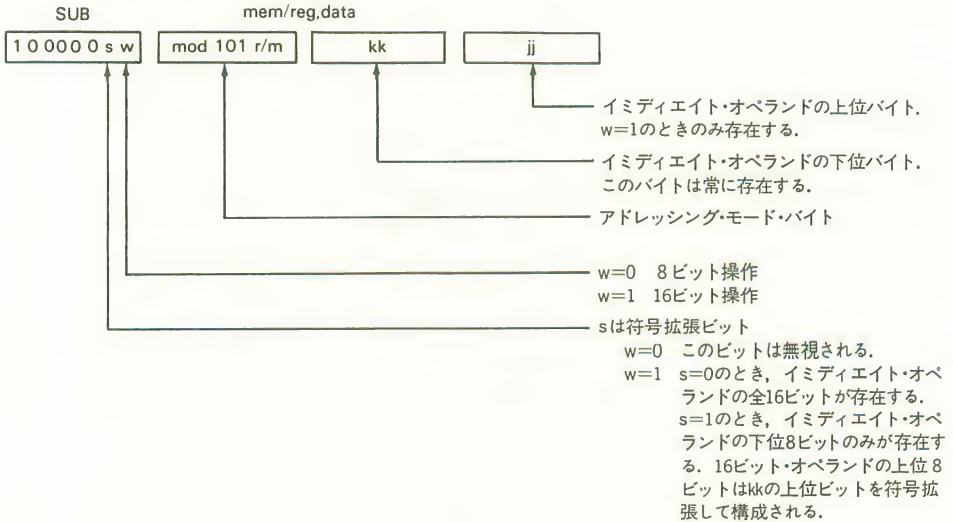
## SUB mem/reg, data (Subtract)

レジスタあるいはメモリからイミディエイト・データを減じる。

指定されたレジスタあるいはメモリ位置から、後続のプログラム・メモリ・バイトのイ

ミディエイト・データを減じる。8あるいは16ビットの操作が指定できる。

命令コードを次に示す。

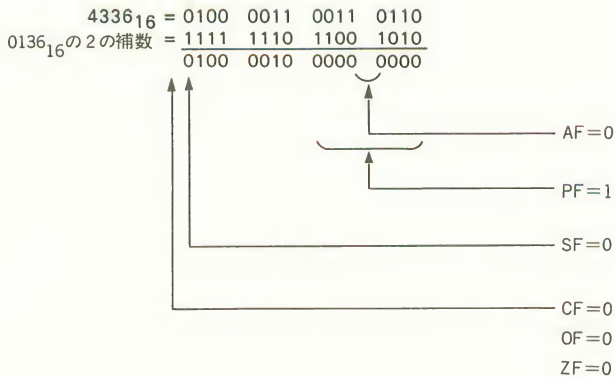


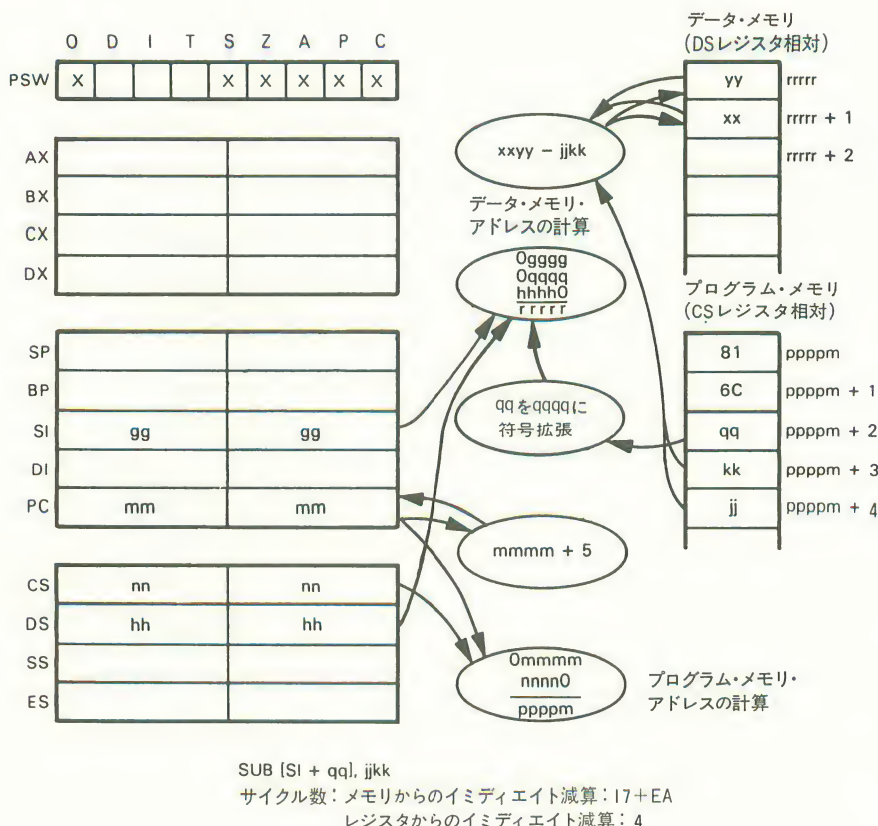
D Sレジスタが  $3000_{16}$  を含み、S Iレジスタが  $0040_{16}$  を含み、メモリ位置  $30054_{16}$  のワードが  $4336_{16}$  であると仮定する。

SUB [SI+14H], 0136H

の実行後、メモリ位置  $30054_{16}$  のワードは  $4200_{16}$  になる。

フラグは以下のように設定される。





注)

- この命令は、普通AXあるいはALレジスタからイミディエイト・データを減じるためには用いられない。この目的のためには、命令 SUB ac,dataがある。

## SUB mem/reg<sub>1</sub>, mem/reg<sub>2</sub> (Subtract)

レジスタからレジスタ  
 メモリからレジスタ  
 レジスタからメモリ

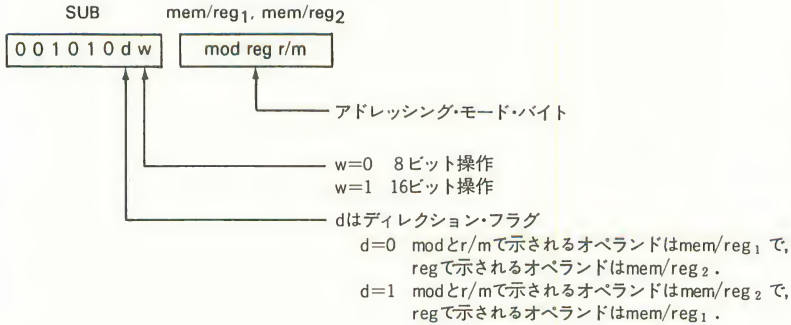
を減じる。

mem/reg<sub>1</sub>で示されるレジスタあるいはメモリ位置の内容から、mem/reg<sub>2</sub>で示されるレジスタあるいはメモリ位置の内容を減じる。8あるいは16ビットの操作が指定できる。mem/reg<sub>1</sub>あるいはmem/reg<sub>2</sub>はメモリ・オペランドとなるが、オペランドの一方はレジス



タ・オペランドでなければならない。

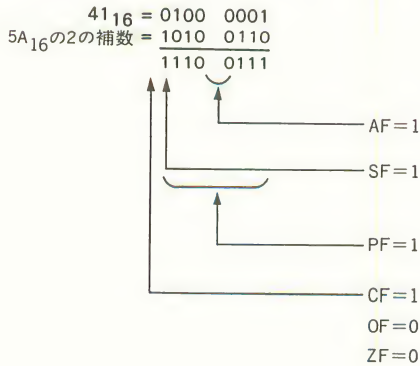
命令コードを次に示す。

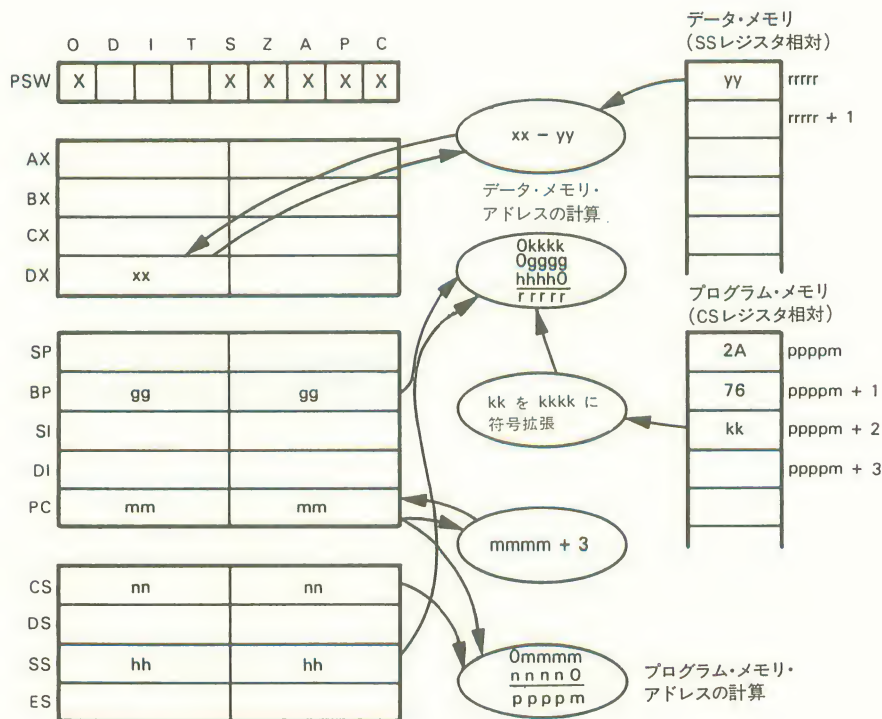


DHレジスタが $41_{16}$ を含み、SSレジスタが $0000_{16}$ を含み、BPレジスタが $00E4_{16}$ を含み、メモリ位置 $000E8_{16}$ のバイトが $5A_{16}$ であるとする。

SUB DH, [BP+4]

の実行後、DHレジスタは $E7_{16}$ になり、ステータスは以下のように設定される。





SUB DH,[BP + kk]

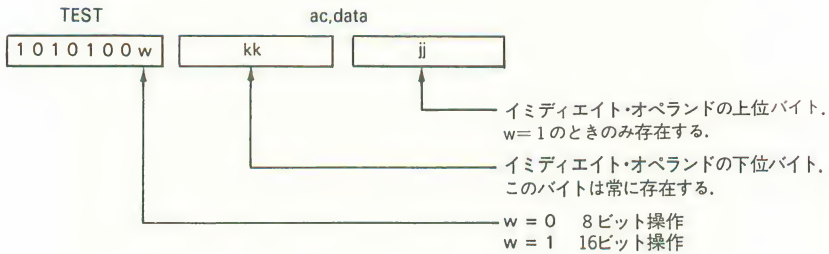
サイクル数: レジスタからメモリを減算 : 9 + EA  
メモリからレジスタを減算 : 16 + EA  
レジスタからレジスタを減算: 3

## TEST ac, data (Test)

AXあるいはALレジスタとイミディエイト・データをテストする。

AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容と、後続のプログラム・メモリ・バイトのイミディエイト・データとのANDをとる。ただし、結果はレジスタに返されない。

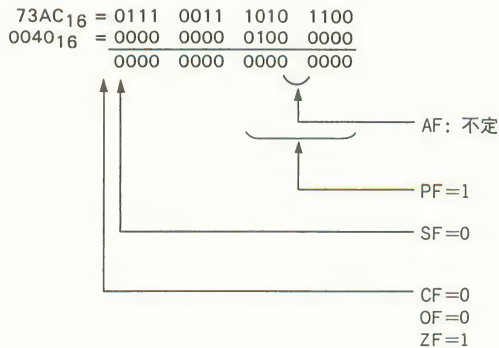
命令コードを次に示す。

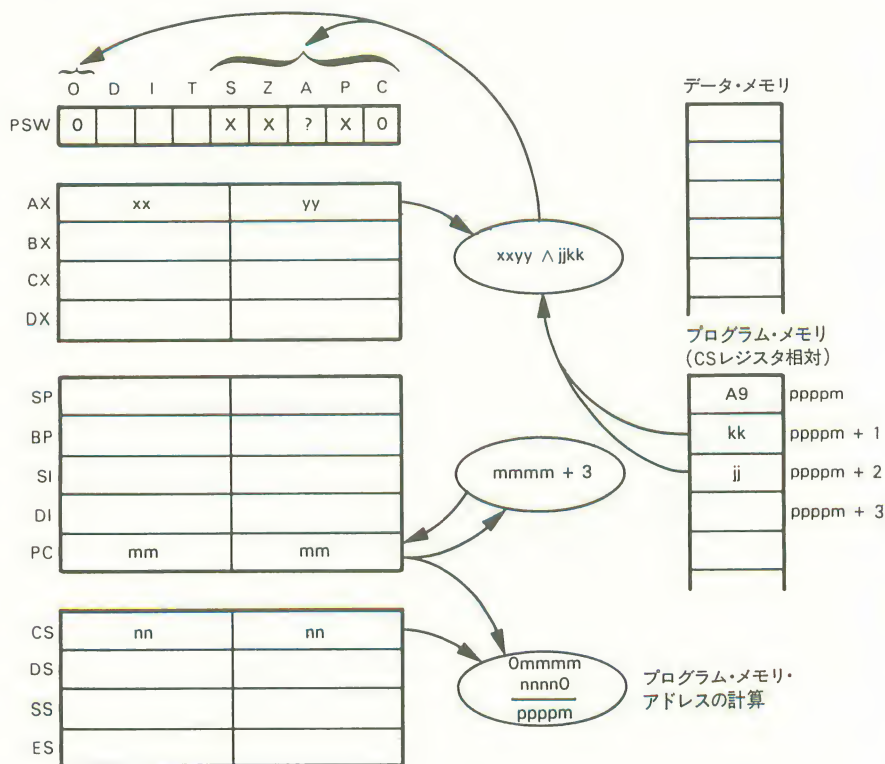


例として、AXレジスタが $73AC_{16}$ を含む場合を考える。

TEST AX,0040H

の実行後、AXレジスタは $73AC_{16}$ のままであるが、フラグ・レジスタは $73AC_{16}$ と $0040_{16}$ のANDによって変化している。





TEST AX,jkk

サイクル数: 4

注)

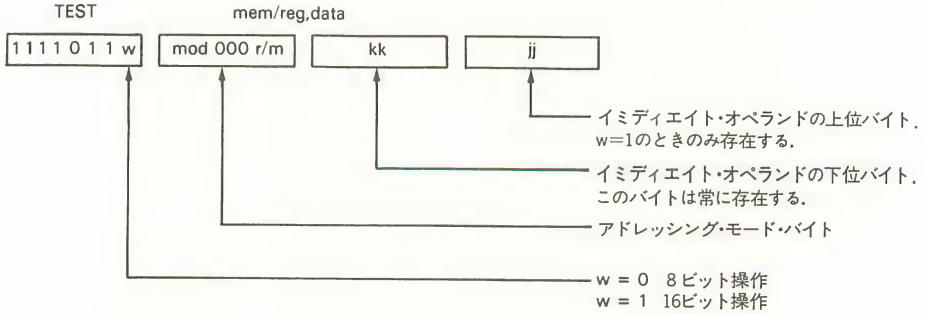
1. 他のレジスタあるいはメモリ位置の内容のTESTが必要ならば, TEST mem/reg, data の命令を参照.

## TEST mem/reg, data (Test)

レジスタあるいはメモリの内容とイミディエイト・データをテストする.

指定されたレジスタあるいはメモリ位置の内容と, 後続のプログラム・メモリ・バイトのイミディエイト・データとのANDをとるが, 結果はレジスタあるいはメモリに返されない. 8あるいは16ビットの操作が指定できる.

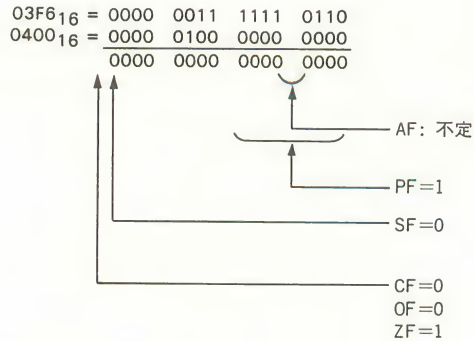
命令コードを次に示す。

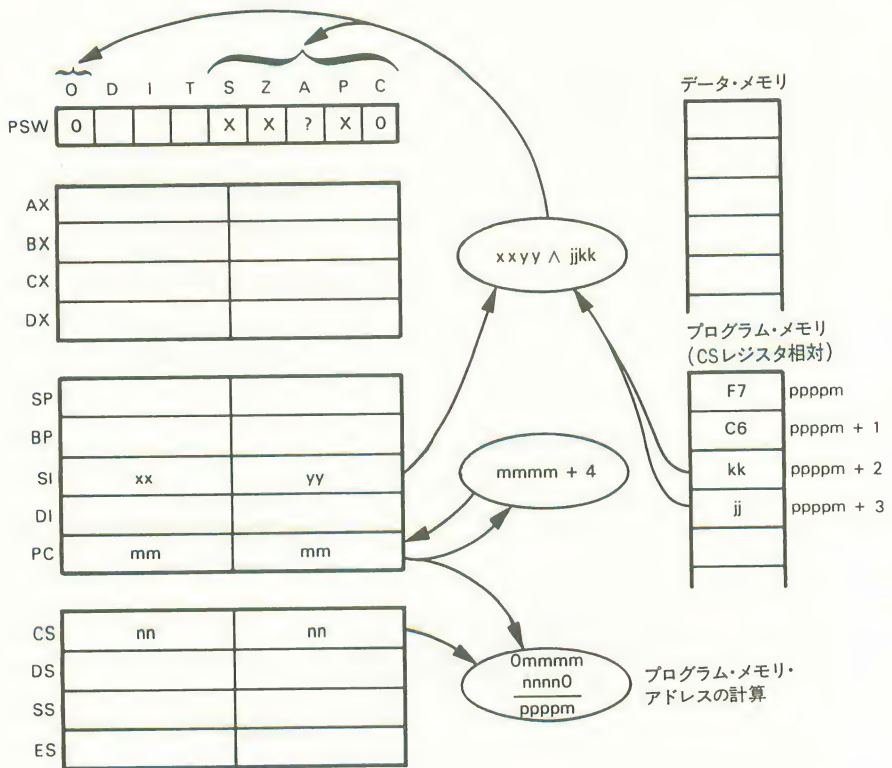


たとえば、SIレジスタが03F6<sub>16</sub>を含む場合を考える。

TEST SI,0400H

の実行後、SIレジスタの内容は変化しないが、フラグは03F6<sub>16</sub>と0400<sub>16</sub>のANDの結果によって設定される。





TEST SI, jjkk

サイクル数: メモリとのイミディエイト: 5

レジスタとのイミディエイト: 11 + EA

注)

1. AXあるいはALのレジスタのテストの目的のためには TEST ac, data があるので、それは通常この命令に関連した機能ではない。

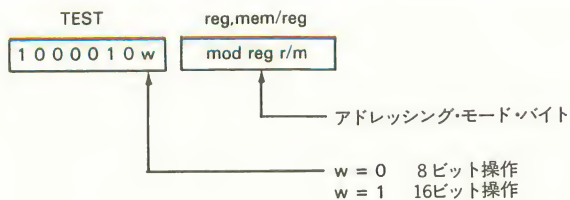
## TEST reg, mem/reg (Test)

メモリとレジスタをテストする。

指定されたレジスタあるいはメモリ位置の内容と、指定されたレジスタの内容とのANDをとり、その結果をフラグに設定するが、結果はレジスタあるいはメモリに返されない。8あるいは16ビットの操作ができる。

命令コードを次に示す。

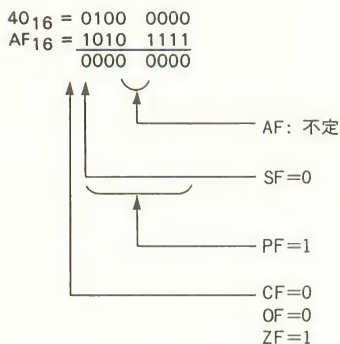


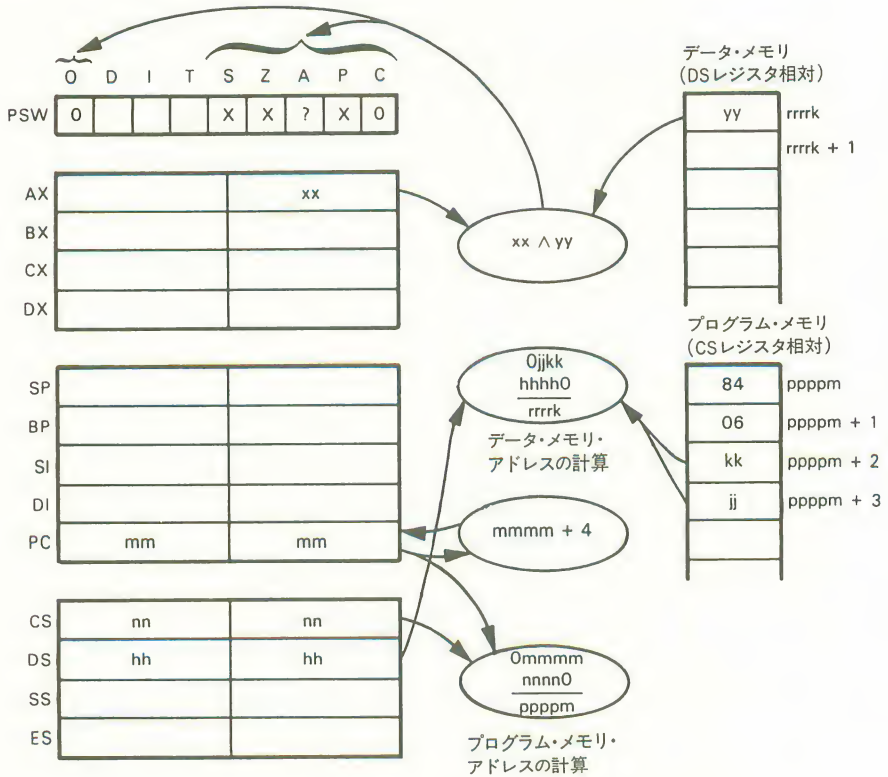


A Lレジスタが  $40_{16}$  を含み、D Sレジスタが  $8800_{16}$  を含み、メモリ位置  $88053_{16}$  のバイトが  $AF_{16}$  であるとする。

TEST AL, [53H]

の実行後、A Lレジスタとメモリ位置  $88053_{16}$  のバイトはどちらも影響を受けないが、フラグは次のように変化する。





TEST AL, [kk]

サイクル数: レジスタとメモリ: 9 + EA  
レジスタとレジスタ: 3

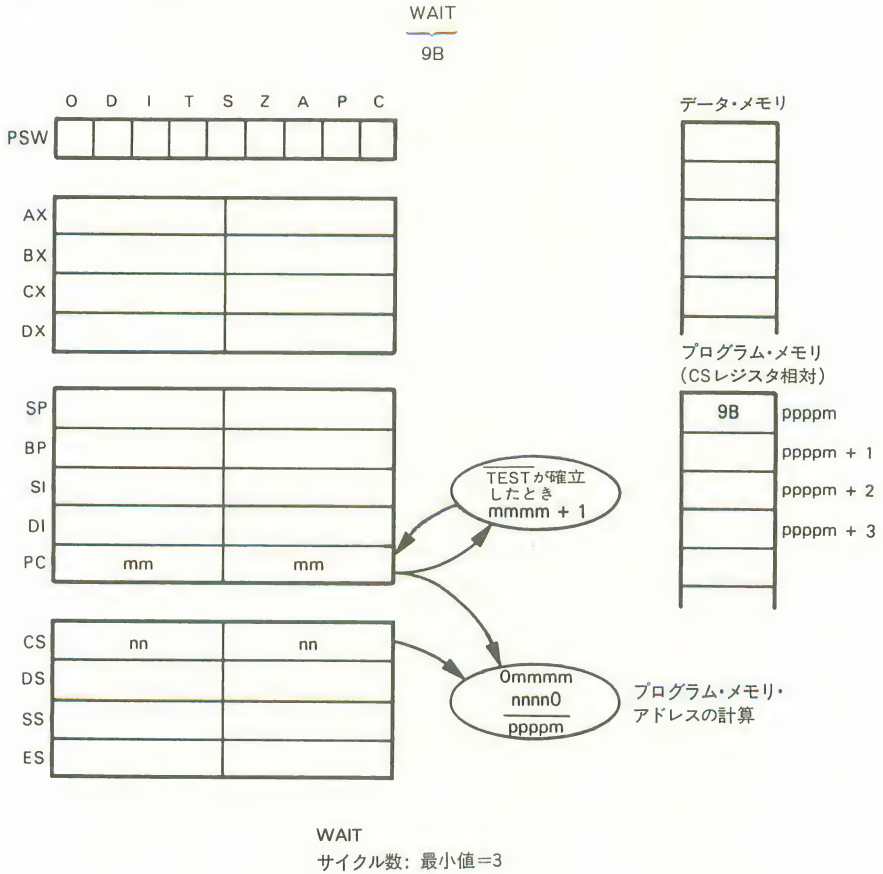
## WAIT (Wait)

テスト・ピンの信号が確立するのを待つ。

TESTピンの信号が確立していなければ、この命令によって8086はアイドル状態となる。8086がアイドル状態から抜け出すのは、以下の2つの条件の1つが成立したときである。

1. インタラプトが有効ならば、外部インタラプトによって8086はインタラプトの処理を行なう。8086がインタラプトの処理を行なうときにセーブされるアドレスは、WAIT命令のアドレスである。したがって、インタラプト・サービス・ルーチンから復帰すると、WAIT命令に戻る。
2. TEST信号が確立している。

命令コードを次に示す。

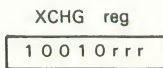


## XCHG reg (Exchange)

アキュムレータとレジスタの内容を交換する。

アキュムレータの内容と、指定された16ビットの内容を交換する。

命令コードを次に示す。



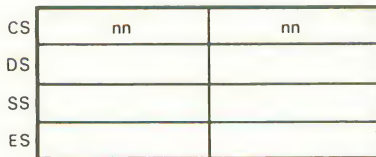
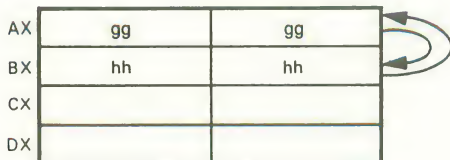
3ビットで、AXレジスタと内容が交換される16ビットのレジスタを指定.

rrr = 000:AX  
 001:CX  
 010:DX  
 011:BX  
 100:SP  
 101:BP  
 110:SI  
 111:DI

たとえば,

XCHG BX

の命令は、BXレジスタの内容をAXレジスタの内容と交換するために用いられる.



mmmm + 1

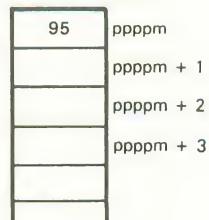
0mmmm  
 nnnn0  
 ppppm

プログラム・メモリ・  
 アドレスの計算

データ・メモリ



プログラム・メモリ  
 (CSレジスタ相対)



XCHG BX  
 サイクル数: 3

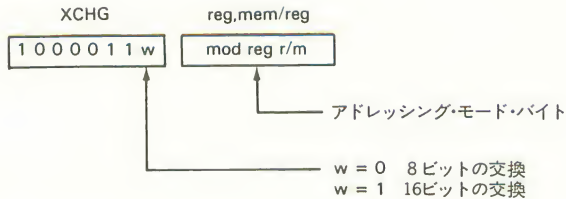
注)

1. ステータスは影響を受けない。
2. 命令 `XCHG AX` は、8086で `NOP` 命令として用いられる。

## `XCHG reg, mem/reg` (Exchange)

レジスタあるいはメモリの内容とレジスタの内容を交換する。

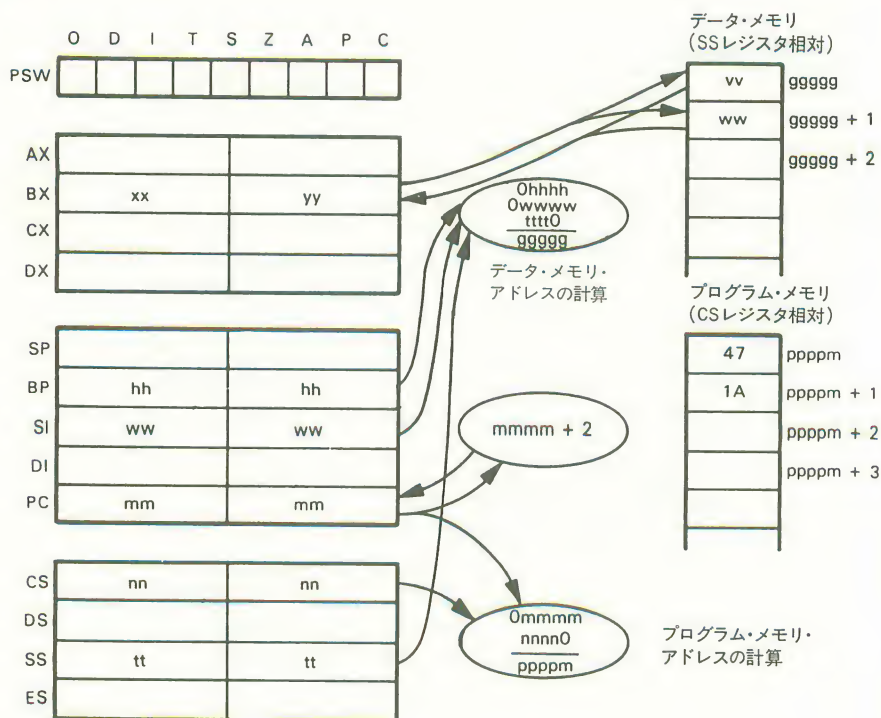
この命令は、`mem/reg` オペランドで示されるレジスタあるいはメモリ位置の内容を、`reg` オペランドで示されるレジスタの内容と交換する。8あるいは16ビットの転送が指定できる。命令コードを次に示す。



`BX` レジスタの内容が  $6F30_{16}$  で、`SS` レジスタが  $2F00_{16}$  を含み、`SI` レジスタが  $0046_{16}$  を含み、`BP` レジスタが  $0200_{16}$  を含み、メモリ位置  $2F246_{16}$  のワードが  $4154_{16}$  の場合を考える。

`XCHG BX, [BP+SI]`

の実行後、`BX` レジスタは  $4154_{16}$  になり、メモリ位置  $2F246_{16}$  は  $30_{16}$  に、メモリ位置  $2F247_{16}$  は  $6F_{16}$  になる。



XCHG BX, [BP + SI]

サイクル数: メモリとレジスタ: 17 + EA

レジスタとレジスタ: 4

注)

1. ステータスは影響を受けない。
2. セグメント・レジスタはこの命令で指定できない。セグメント・レジスタの内容を交換する命令は存在しない。
3. 普通、この命令はレジスタとAXレジスタの交換には用いられない。このためには、命令 XCHG reg がある。

## XLAT (Translate)

ALとBXのレジスタによってテーブルを検索する。

8ビットのデータがALレジスタにロードされる。このデータ要素のアドレスは、次のアルゴリズムを用いて求められる。

1. 16ビットのBXレジスタに、ALレジスタの8ビットの内容を加算する。



2. ステップ1の加算結果を、DSレジスタに対するオフセット・アドレスとして用いる(セグメント変更が行なわれていないと仮定)。

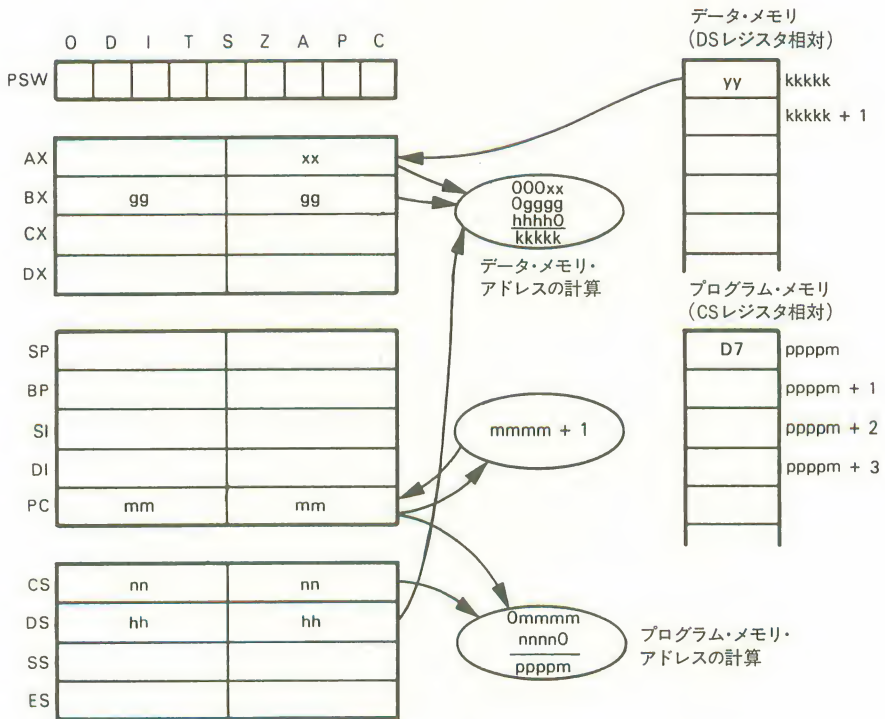
XLATの命令コードを次に示す。

XLAT  
——  
D7

たとえば、ALレジスタが $0F_{16}$ を含み、BXレジスタが $0040_{16}$ 、DSレジスタが $F000_{16}$ とすると、

XLAT

の実行によって、メモリ位置 $F004F_{16}$ の内容がALレジスタにロードされる。



XLAT  
サイクル数: 11

注)

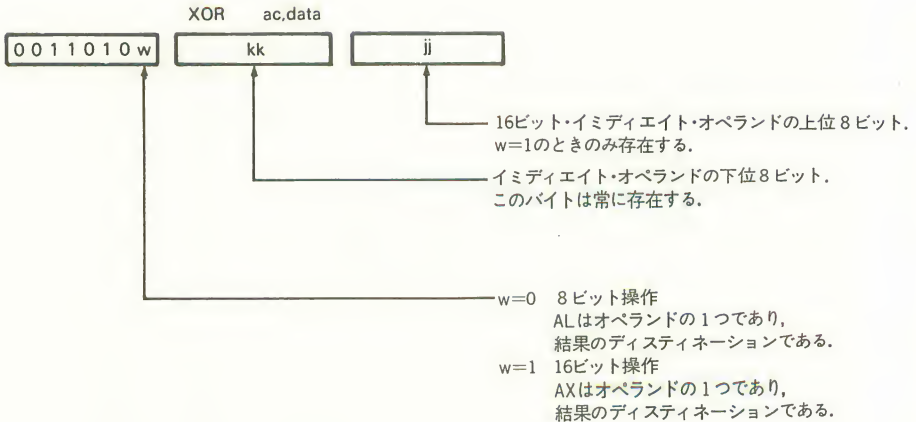
1. この命令は、BXレジスタがテーブルの開始アドレスを含み、ALレジスタをテーブルのインデックスとして利用する場合に用いられるのが最も一般的である。

## XOR ac, data (Exclusive-OR)

A XあるいはA Lのレジスタとイミディエイト・データのX O Rをとる。

この命令は、イミディエイト・アドレッシングによって、A L（8ビット）あるいはA X（16ビット）のレジスタと、8あるいは16ビットのデータとのエクスクルーシブO Rをとる。

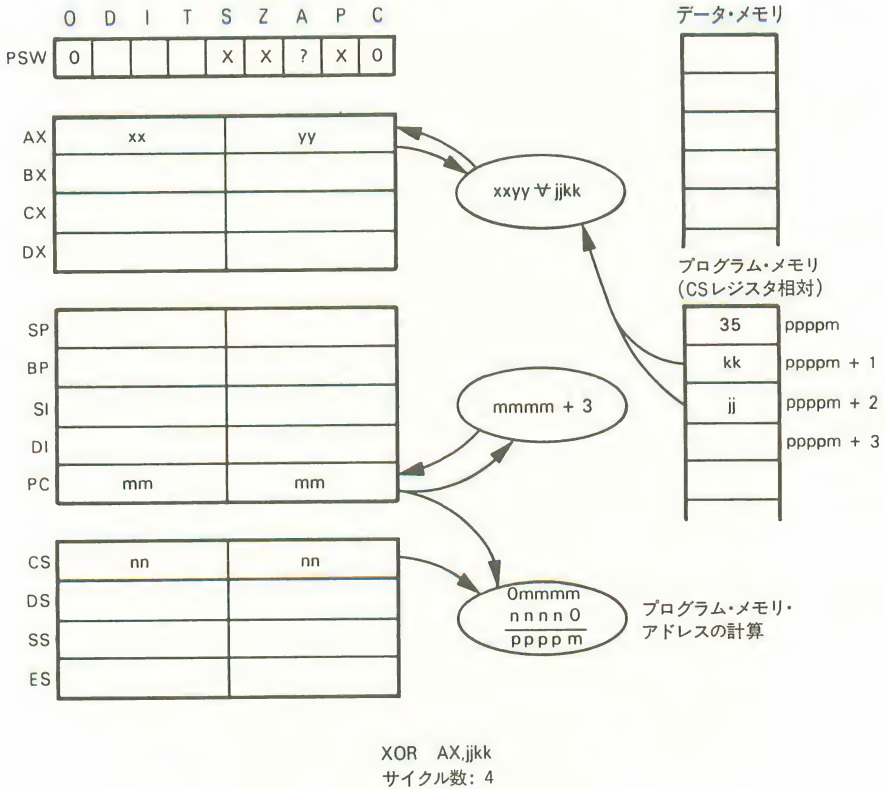
命令コードを次に示す。



たとえば、A Xが  $B31C_{16}$  を含むとする。

XOR AX, 5522H

の実行によって、A Xレジスタには  $E63E_{16}$  がストアされる。



注)

1. この命令は、8080アセンブリ言語において、XRI data 命令と同じ機能を果たす。ただし、この命令は16ビットのデータも可能であるが、8080のXRIは8ビットのデータ要素のみを用いる。

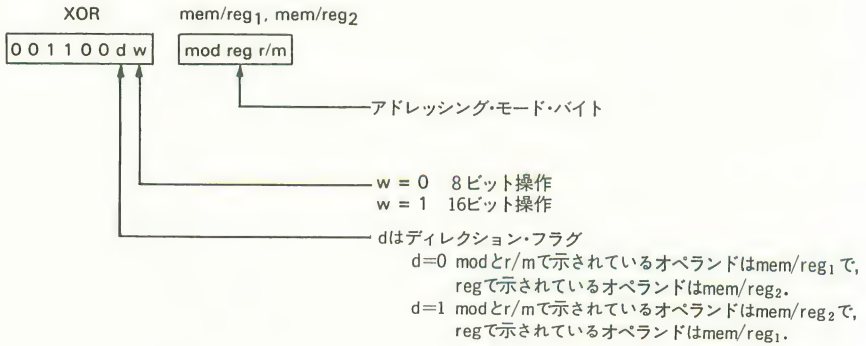
## XOR mem/reg<sub>1</sub>, mem/reg<sub>2</sub> (Exclusive-OR)

レジスタとレジスタ  
レジスタとメモリ  
メモリとレジスタ

でXORをとる。

mem/reg<sub>2</sub>で示されるレジスタあるいはメモリ位置の内容と、mem/reg<sub>1</sub>で示されるレジスタあるいはメモリ位置の内容のエクスクルーシブORを取り、結果をmem/reg<sub>1</sub>に返す。8あるいは16ビットの操作が指定できる。mem/reg<sub>1</sub>あるいはmem/reg<sub>2</sub>はメモリ・オペランドとなるが、オペランドの一方はレジスタ・オペランドでなければならない。

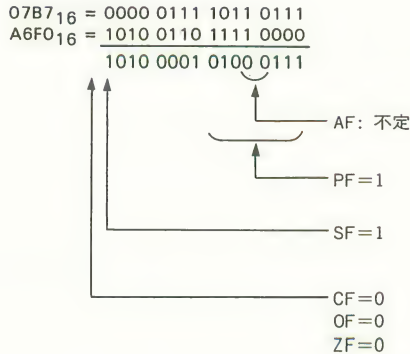
命令コードを次に示す。

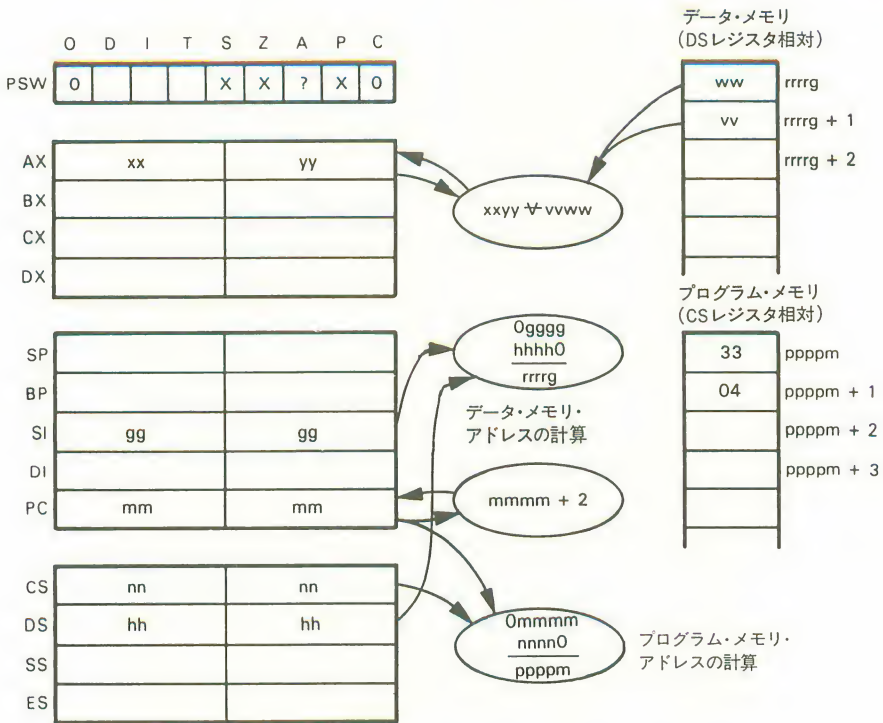


A Xレジスタが  $07B7_{16}$  を含み, D Sレジスタが  $9080_{16}$  を含み, S Iレジスタが  $040E_{16}$  を含み, メモリ位置  $90C0E_{16}$  のワードが  $A6F0_{16}$  であるとする。

XOR AX,[SI]

の実行後, A Xレジスタは  $A147_{16}$  になる。フラグは以下のように設定される。





XOR AX,[SI]

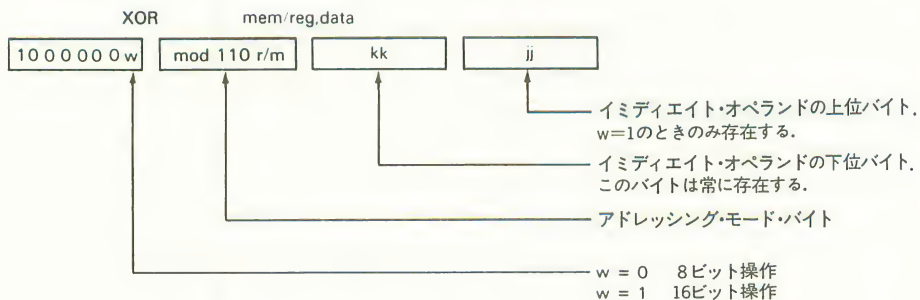
サイクル数: メモリからレジスタに対して: 9 + EA  
 レジスタからメモリに対して: 16 + EA  
 レジスタからレジスタに対して: 3

## XOR mem/reg, data (Exclusive-OR)

レジスタあるいはメモリの内容とイミディエイト・データのXORをとる。

指定されたレジスタあるいはメモリ位置の内容と、後続のプログラム・メモリ・バイトのイミディエイト・データとのXORをとる。8あるいは16ビットの操作が指定できる。

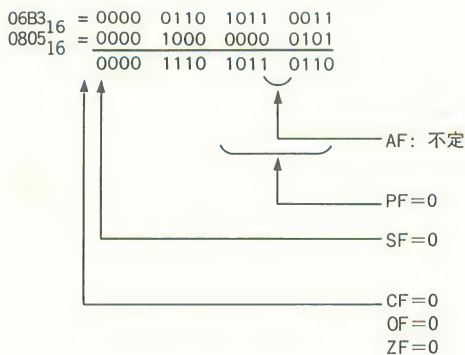
命令コードを次に示す.



D S レジスタが  $3800_{16}$  を含み, B X レジスタの内容が  $0200_{16}$  で, D I レジスタが  $0136_{16}$  を含み, メモリ位置  $38336_{16}$  のワードが  $06B3_{16}$  の場合を考える.

XOR    [BX+DI], 0805H

の実行後, メモリ位置  $38336_{16}$  のワードは  $0EB6_{16}$  になる.







### 3.7 アセンブラ依存のニーモニック

8086アセンブラのいくつかのニーモニックは、8あるいは16ビットのいずれの操作が行なわれるべきかは明確に定義しない。オペランドとして8086のレジスタを持つすべての命令は、8ビットあるいは16ビットのどちらの操作が必要であるかを決めるのに、レジスタを用いることができる。たとえば、

```
XOR AX,0804H
```

の命令が16ビット操作であることは明らかである。しかし、命令の中にはレジスタを指定しないが8ビットあるいは16ビットの操作が可能なものがある。このような命令には、CMP SやLODSなどのストリング操作と、MULやNOTなどの単一メモリ・オペランドが指定できる命令の、2つの基本的な形式が含まれている。

アセンブラはこの問題を以下の3つの方法で処理している。

1. ワードあるいはバイトの操作のどちらを指定するかを表わす文字を、ニーモニックに付加することができる。たとえば、MULは次のように指定できる。

```
MULB 8×8ビットの乗算
```

```
MULW 16×16ビットの乗算
```

ストリング操作の場合、文字はニーモニックの最後のB（バイト）あるいはW（ワード）である。たとえば、命令CMP Sは次のもので置き換えられる。

```
CMPSB 8ビットの比較
```

```
CMPSW 16ビットの比較
```

2. ストリング操作のオペランドは、WORDあるいはBYTEである。たとえば、MOV S命令は次のようになる。

```
MOVB BYTE 8ビットの移動
```

```
MOVW WORD 16ビットの移動
```

さらに、この方法は単一メモリ・オペランドへ適用できる。次に例を示す。

```
NOT SI,BYTE
```

```
NOT SI,WORD
```

3. プログラマは、すべてのデータ領域とシンボルを、WORDあるいはBYTEの要素として定義する。アセンブラはこの情報を保持していて、データ領域やシンボルへの参照が行なわれたときに、8あるいは16ビットの操作のどちらが必要かを決定する。たとえば、

```
TOUCH$TONE$OUTPUT$BYTE DB 00H ; DB means define byte
```

```
TIMER DW 0000H ; DW means define word
```

このとき、以下の操作が行なわれるならば、

```
NOT TOUCH$TONE$OUTPUT$BYTE
```

```
INC TIMER
```

NOT操作は8ビットの操作としてアセンブルされ、INC操作には16ビットの操作

が指定される。特定のアセンブラは、その機能を果たすために、これらのオプションのうちの1つ以上が用いられることに注意。

ストリング命令について、オブジェクト・コード・レベルの命令はオペランドを含んでいないが、命令の汎用のソース形式ではオペランドを指定する必要がある。この条件は、アセンブラにバイトあるいはワードのどちらの操作が必要かの決定を可能とする。たとえば、

#### LODS

の命令を考える。ソース・オペランドのアドレスはDSとSIのレジスタで指定され、ディスティネーションはAXあるいはALのレジスタとなる。ニーモニックLODSだけでは、オペランドの型（バイトあるいはワード）を決定するのに十分な情報は得られない。しかし、

```

.ID_NUMBER      DB 5
-
-
-
LODS ID_NUMBER

```

によって、LODSからバイト・ロードの形式をアセンブラが自動的に構成できるデータの型の情報が得られる。この表記法はまた、プログラマがロード内容を指定できるので、命令コードの読みやすさと保守性を高めている。



# 第4章

## 8086の命令グループ

この章には、8086命令セットについて別の解説が含まれている。すなわち各命令が個々に述べられている3章の説明に対して、この章では命令のグループについて述べる。8086の命令は、実行される機能によって次のグループに分かれる。

- データ移動命令
- 算術演算命令
- 論理演算命令
- ストリング・プリミティブ命令
- プログラム・カウンタ制御命令
- プロセッサ制御命令
- 入出力命令
- インタラプト命令
- ローテートとシフトの命令

### 4.1 データ移動命令

データの移動を行なう8086の命令を表4-1に示す。8086のデータ移動命令は、以下の3つの一般的部類に分けられる。

1. レジスタからレジスタへ、あるいはメモリ位置とレジスタの間で、データを移動する命令。
2. スタックに対して、データを移動する命令。
3. あるメモリ位置から他へ、複数のバイトを移動する命令。

この第1と第2の命令のタイプについては、この節で述べる。ストリング・プリミティブと呼ばれる命令を用いて構成される複数バイトの移動命令については、この節では外面だけを述べる。その詳細は本章の後で述べる。

データ移動命令は、次のタイプのルーチンで用いられる。

表 4-1 8086のデータ移動命令

| ニーモニック | オペランド                                          | オブジェクト・コード                                                       | バイト           | クロック                                                              | ステータス |   |   |   |   |   |   |   | 動作内容                                                                                                                                             |
|--------|------------------------------------------------|------------------------------------------------------------------|---------------|-------------------------------------------------------------------|-------|---|---|---|---|---|---|---|--------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                                                |                                                                  |               |                                                                   | O     | D | I | T | S | Z | A | P | C                                                                                                                                                |
| MOV    | mem/reg <sub>1</sub> ,<br>mem/reg <sub>2</sub> | 100010dw<br>mod rrr/r/m<br>(DISP)<br>(DISP)                      | 2, 3 または 4    | reg - reg: 2<br>mem - reg: 8 + EA<br>reg - mem: 9 + EA<br>10 + EA |       |   |   |   |   |   |   |   | [mem/reg] ← [mem/reg <sub>2</sub> ]<br>8あるいは16ビットのデータ要素を、mem/reg <sub>2</sub> で示されるメモリ位置あるいはレジスタから、mem/reg <sub>1</sub> で示されるメモリ位置あるいはレジスタに移動する。 |
| MOV    | mem/reg,<br>data                               | 1100011w<br>mod 000 r/m<br>(DISP)<br>(DISP)<br>kk<br>ij (w=1のとき) | 3, 4, 5 または 6 |                                                                   |       |   |   |   |   |   |   |   | [mem/reg] ← data<br>8あるいは16ビットのイミディエイトデータを、mem/regで示されるメモリ位置あるいはレジスタに移動する。                                                                       |
| MOV    | reg, data                                      | 1011wrr<br>kk<br>ij (w=1のとき)                                     | 2 または 3       | 4                                                                 |       |   |   |   |   |   |   |   | [reg] ← data<br>8あるいは16ビットのイミディエイトデータを、regで示されるレジスタに移動する。                                                                                        |
| MOV    | ac, mem                                        | 1010000w<br>kk<br>ij                                             | 3             | 10                                                                |       |   |   |   |   |   |   |   | [ac] ← [mem]<br>memで示されるメモリ位置からデータを、AL (8ビット操作) あるいはAX (16ビット操作) のレジスタに移動する。                                                                     |
| MOV    | mem, ac                                        | 1010001w<br>kk<br>ij                                             | 3             | 10                                                                |       |   |   |   |   |   |   |   | [mem] ← [ac]<br>AL (8ビット操作) あるいはAX (16ビット操作) のレジスタからデータを、memで示されるメモリ位置に移動する。                                                                     |
| MOV    | seg, reg,<br>mem/reg                           | 8E<br>mod 0 ss/r/m<br>(DISP)<br>(DISP)                           | 2, 3 または 4    | reg - reg: 2<br>mem - reg: 8 + EA                                 |       |   |   |   |   |   |   |   | [seg, reg] ← [mem/reg]<br>mem/regで示されるメモリ位置あるいはレジスタから16ビットのデータを選択されたセグメント・レジスタに移動する。ss=01のときは定数されない。                                             |
| MOV    | mem/reg,<br>seg, reg                           | 8C<br>mod 0 ss/r/m<br>(DISP)<br>(DISP)                           | 2, 3 または 4    | reg - reg: 2<br>mem - reg: 9 + EA                                 |       |   |   |   |   |   |   |   | [mem/reg] ← [seg, reg]<br>選ばれたセグメント・レジスタの内容を、指定されたメモリ位置あるいはレジスタに移動する。                                                                            |
| XCHG   | reg,<br>mem/reg                                | 1000011w<br>mod rrr/r/m<br>(DISP)<br>(DISP)                      | 2, 3 または 4    | reg - reg: 4<br>reg - mem: 17 + EA                                |       |   |   |   |   |   |   |   | [reg] ↔ [mem/reg]<br>regで示されるレジスタの8あるいは16ビットの内容を、mem/regで示されるメモリ位置あるいはレジスタの内容と交換する。                                                              |

\* mem ← memは含まれない。



表4-1 8086のデータ移動命令(続き)

| ニーモニック | オペランド    | オブジェクト・コード                            | バイト      | クロック                       | ステータス |   |   |   |   |   |   |   |   |  | 動作内容                                                                                                                                    |
|--------|----------|---------------------------------------|----------|----------------------------|-------|---|---|---|---|---|---|---|---|--|-----------------------------------------------------------------------------------------------------------------------------------------|
|        |          |                                       |          |                            | O     | D | I | T | S | Z | A | P | C |  |                                                                                                                                         |
| XCHG   | reg      | 10010rrr                              | 1        | 3                          |       |   |   |   |   |   |   |   |   |  | $[AX] \leftrightarrow [reg]$<br>AXレジスタの内容と指定されたレジスタの内容とを交換する。                                                                           |
| XLAT   |          | D7                                    | 1        | 11                         |       |   |   |   |   |   |   |   |   |  | $[AL] \leftarrow [[AL] + [BX]]$<br>ALとBXの和で示されるデータ・バイトをALレジスタにロードする。                                                                    |
| LDS    | reg, mem | C5<br>mod rrr r/m<br>(DISP)<br>(DISP) | 2, 3または4 | 16 + EA                    |       |   |   |   |   |   |   |   |   |  | $[reg] \leftarrow [mem], [DS] \leftarrow [mem + 2]$<br>memで示されるメモリ位置から16ビットのデータを選択されたレジスタにロードする。memで示されるメモリ位置の次の16ビットのデータをDSレジスタにロードする。 |
| LEA    | reg, mem | 8D<br>mod rrr r/m<br>(DISP)<br>(DISP) | 2, 3または4 | 2 + EA                     |       |   |   |   |   |   |   |   |   |  | $[reg] \leftarrow mem$ (アドレスのオフセット部分)<br>メモリ・アドレスのオフセット部分となる16ビットを選択されたレジスタにロードする。                                                      |
| LES    | reg, mem | C4<br>mod rrr r/m<br>(DISP)<br>(DISP) | 2, 3または4 | 16 + EA                    |       |   |   |   |   |   |   |   |   |  | $[reg] \leftarrow [mem], [ES] \leftarrow [mem + 2]$<br>memで示されるメモリ位置から16ビットのデータを選択されたレジスタにロードする。memで示されるメモリ位置の次の16ビットのデータをESレジスタにロードする。 |
| PUSH   | mem/reg  | FF<br>mod 110 r/m<br>(DISP)<br>(DISP) | 2, 3または4 | reg: 11<br>mem:<br>16 + EA |       |   |   |   |   |   |   |   |   |  | $[SP] \leftarrow [SP] - 2, ([SP]) \leftarrow [mem/reg]$<br>SPを2だけ減じる。mem/regで示されるメモリ位置あるいはレジスタの16ビットの内容をスタックのトップにストアする。                 |
| PUSH   | reg      | 01010rrr                              | 1        | 10                         |       |   |   |   |   |   |   |   |   |  | $[SP] \leftarrow [SP] - 2, ([SP]) \leftarrow [reg]$<br>SPを2だけ減じる。指定されたレジスタの16ビットの内容をスタックのトップにストアする。                                     |
| PUSH   | seg, reg | 000ss110                              | 1        | 10                         |       |   |   |   |   |   |   |   |   |  | $[SP] \leftarrow [SP] - 2, ([SP]) \leftarrow [seg, reg]$<br>SPを2だけ減じる。指定されたセグメント・レジスタの16ビットの内容をスタックのトップにストアする。                          |
| PUSHF  |          | 9C                                    | 1        | 10                         |       |   |   |   |   |   |   |   |   |  | $[SP] \leftarrow [SP] - 2, ([SP]) \leftarrow [FLAGS]$<br>SPを2だけ減じる。フラグ・レジスタの内容をスタックのトップにストアする。                                          |

表 4-1 8086のデータ移動命令 (続き)

| ニーモニック | オペランド   | オブジェクト・コード                            | バイト        | クロック                      | ステータス |   |   |   |   |   |   |   | 動作内容 |                                                                                                                                                                                                          |
|--------|---------|---------------------------------------|------------|---------------------------|-------|---|---|---|---|---|---|---|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |         |                                       |            |                           | O     | D | I | T | S | Z | A | P |      | C                                                                                                                                                                                                        |
| POP    | mem/reg | 8F<br>mod 000 r/m<br>(DISP)<br>(DISP) | 2, 3 または 4 | reg: 8<br>mem:<br>17 + EA |       |   |   |   |   |   |   |   |      | $\{mem/reg\} \leftarrow \{[SP]\}, \{[SP]\} \leftarrow \{[SP]\} + 2$<br>スタックのトップの16ビットを, mem/regで示されるメモリ位置あるいはレジスタに移動する. SPを2だけ増やす.                                                                       |
| POP    | reg     | 01011rr                               | 1          | 8                         |       |   |   |   |   |   |   |   |      | $\{reg\} \leftarrow \{[SP]\}, \{[SP]\} \leftarrow \{[SP]\} + 2$<br>スタックのトップの16ビットを, 選択されたレジスタに移動する. SPを2だけ増やす.                                                                                           |
| POP    | segreg  | 000ss111                              | 1          | 8                         |       |   |   |   |   |   |   |   |      | $\{segreg\} \leftarrow \{[SP]\}, \{[SP]\} \leftarrow \{[SP]\} + 2$<br>スタックのトップの16ビットを, 指定されたセグメント・レジスタに移動する. SPを2だけ増やす.<br>ss=01のときは定義されない.                                                              |
| POPF   |         | 9D                                    | 1          | 8                         | X     | X | X | X | X | X | X | X | X    | $\{FLAGS\} \leftarrow \{[SP]\}, \{[SP]\} \leftarrow \{[SP]\} + 2$<br>スタックのトップの16ビットをフラグ・レジスタに移動する. SPを2だけ増やす.                                                                                            |
| LAHF   |         | 9F                                    | 1          | 4                         |       |   |   |   |   |   |   |   |      | 8080AのフラグをAHレジスタに移動する.<br><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div>— AH</div><div>S</div><div>Z</div><div>A</div><div>P</div><div>C</div></div> |
| SAHF   |         | 9E                                    | 1          | 4                         |       |   |   |   | X | X | X | X | X    | AHレジスタを8080Aのフラグに移動する.<br><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div>— AH</div><div>S</div><div>Z</div><div>A</div><div>P</div><div>C</div></div> |

1. BUFFER\$Aの内容をBUFFER\$Bに移すルーチン.
2. BUFFER\$Aの内容を初期設定するルーチン.
3. BUFFER\$Aの内容を変換するルーチン.

#### 4.1.1 バッファからバッファへの移動ルーチン

8ビットと16ビットのデータ要素に対して、2つの基本的なバッファからバッファへの移動ルーチンを、それぞれ、図4-1と図4-2に示す。このルーチンでは、SIレジスタがBUFFER\$Aのアドレスを含み、DIレジスタがBUFFER\$Bのアドレスを含み、CXレジスタが移動するデータ要素の数を含むと仮定している。

|              |     |             |                              |
|--------------|-----|-------------|------------------------------|
| MOVE\$BYTES: | MOV | AL,[SI]     | ;LOAD BYTE FROM SOURCE       |
|              | MOV | [DI],AL     | ;STORE BYTE INTO DESTINATION |
|              | INC | SI          | ;ADJUST POINTERS             |
|              | INC | DI          |                              |
|              | DEC | CX          | ;DECREMENT # TO MOVE         |
|              | JNZ | MOVE\$BYTES | ;LOOP IF NOT DONE            |
|              | RET |             |                              |

図4-1 8ビットのバッファからバッファへの移動

|              |     |             |                              |
|--------------|-----|-------------|------------------------------|
| MOVE\$WORDS: | MOV | AX,[SI]     | ;LOAD WORD FROM SOURCE       |
|              | MOV | [DI],AX     | ;STORE WORD INTO DESTINATION |
|              |     |             | ;ADJUST POINTERS             |
|              | INC | SI          |                              |
|              | INC | SI          |                              |
|              | INC | DI          |                              |
|              | INC | DI          |                              |
|              | DEC | CX          | ;DECREMENT # TO MOVE         |
|              | JNZ | MOVE\$WORDS | ;LOOP IF NOT DONE            |
|              | RET |             |                              |

図4-2 16ビットのバッファからバッファへの移動

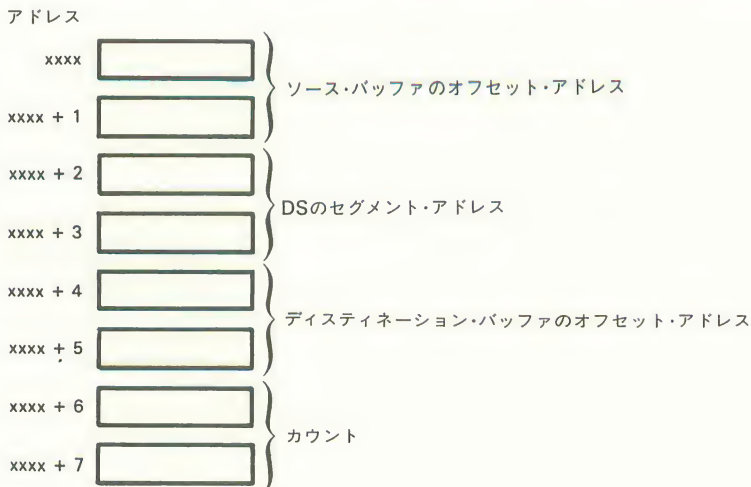
図4-1と図4-2に示した一連の命令は、データ・セグメント内でデータを移動する。このルーチンでは、SIあるいはDIのレジスタの代わりにBXレジスタを用いることができる。

ここに示したルーチンは容易に理解できるが、あまり能率的ではない。この章で後述述べるストリング・プリミティブ操作は、より効力のあるバッファからバッファへの移動ルーチンを構成する。また、LOOP命令は、デクリメントと分岐によるプログラム・ロジックを能率的なものにする。

#### (1) バッファからバッファへの移動のためのレジスタの初期設定

バッファからバッファへの移動ルーチンなどの処理で用いられるレジスタの初期設定には、多くの方法がある。初期設定の方法は、アドレスとカウントがどのように得られるかにかかっている。付加的要素には、初期設定されるべきレジスタの数がある。たとえば、

多くの場合、DSレジスタは既に初期設定されている。バッファからバッファへの移動ルーチンの、バッファ先頭アドレスとバイトあるいはワードの数は、レジスタで示されるメモリ・ワードのブロックに保持することができる。次の8バイトのメモリ・ブロックを考える。



上に示されているように、対になるメモリ・バイトはアドレスを保持する。

上に示したメモリ・ワードのブロックは、しばしばパラメータ・ブロックと呼ばれる。ブロックの個々のデータの値は、パラメータである。

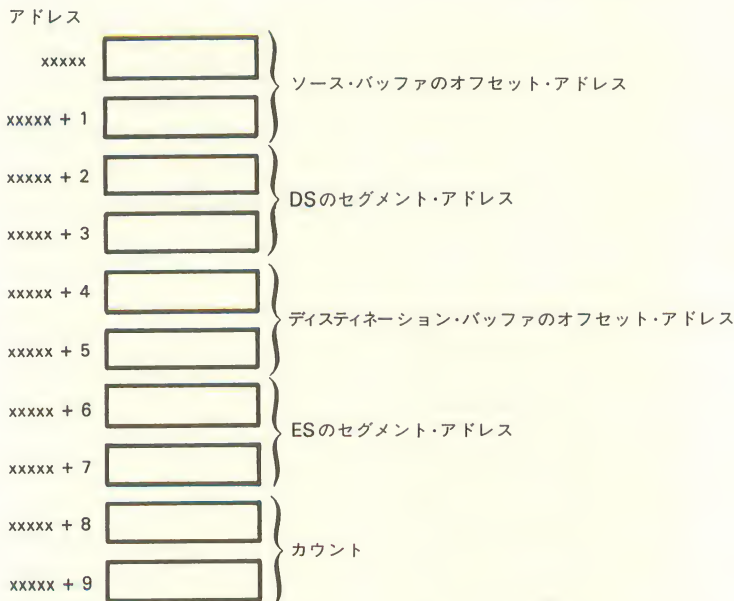
初期設定のパラメータを得るために、DIレジスタはこのブロックの先頭アドレス（上の例ではxxxx）をロードする。

|     |             |
|-----|-------------|
| LDS | SI,[DI]     |
| MOV | CX,[DI + 6] |
| MOV | DI,[DI + 4] |

図 4-3 バッファ移動レジスタ初期設定

もし都合が良ければ、DIレジスタの代わりにBXレジスタを用いることができる。

ストリング・プリミティブ命令を用いるときは、ESレジスタにロードされるセグメント・アドレスを含むため、パラメータ・ブロックは以下に示すように拡張される必要がある。



この方法では、1メガバイトのメモリ領域で、任意の位置から別の位置へデータを移動できる。セグメントが指定されていなければ、カレント・セグメント内だけで、データが移動される。

また、このブロックの先頭アドレス（上の例ではxxxx）をDIレジスタにロードすると、初期設定は次のようになる。

|     |             |
|-----|-------------|
| LDS | SI,[DI]     |
| MOV | CX,[DI + 8] |
| LES | DI,[DI + 4] |

図4-4 別のバッファ移動レジスタ初期設定

1つのバッファが常にメモリの固定位置にあるならば、固定バッファのアドレスはイミディエイト・データとして指定できる。次の一連の命令を考える。

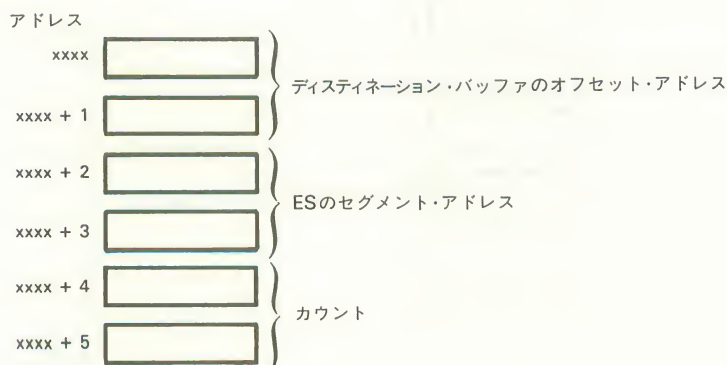
|     |                            |
|-----|----------------------------|
| MOV | SI,ADDR\$FOR\$BUFFER\$A    |
| MOV | AX,SEGADDR\$FOR\$BUFFER\$A |
| MOV | DS,AX                      |
| MOV | CX,[DI + 4]                |
| LES | DI,[DI]                    |

図4-5 イミディエイト・データを用いたバッファ移動レジスタ初期設定

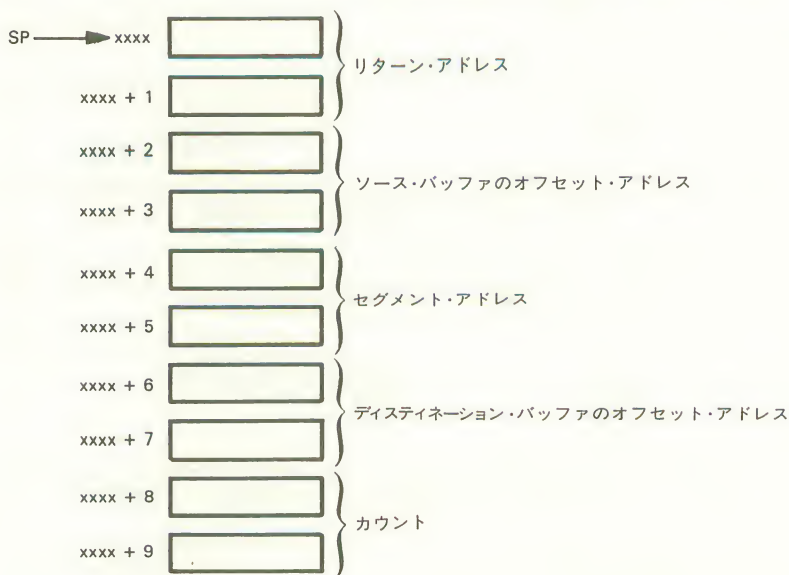
最初の命令では、イミディエイト・データのADDR\$FOR\$BUFFER\$AがSIレジスタに移動される。第2の命令は、イミディエイト・データのSEGADDR\$FOR\$BUFFER\$AをAXレジスタに移動する。この命令は、8086がイミディエイト・データをセグメン



ト・レジスタに移動する命令を持たないので、必要となる（例外に、16ビットのセグメント・アドレスをコード・セグメント・レジスタにロードするセグメント間ジャンプ命令がある）。第3の命令は、セグメント・アドレスをDSレジスタに移動する。しばしばDSレジスタはすでに必要な値に設定されていて、修正の必要がないことに注意。第4と第5の命令は、適当なレジスタにカウントとディスティネーション・バッファ・アドレスをロードするために用いられる。これらの命令では、DIレジスタは次の形式のブロックを示している必要がある。



パラメータ（この場合、アドレスとカウントの情報）は、スタックを通してルーチンに受け渡すことができる。バッファからバッファへの移動に対しては、次のように示される。





次の一連の命令は、レジスタの初期設定を行なう。

|     |    |                    |
|-----|----|--------------------|
| POP | BX | POP RETURN ADDRESS |
| POP | SI |                    |
| POP | DS |                    |
| POP | DI |                    |
| POP | CX |                    |

図4-6 スタックとポップ命令によるバッファ移動レジスタ初期設定

この方法では、RET命令を用いてサブルーチンから復帰することは困難となる。しかし、8086ではレジスタをリターン・アドレスとすることができる。したがって、

JMP BX

をRETの代わりに用いることができる。この方法が本質的に好ましくないか、あるいはすべてのレジスタが使用されているならば、次の方法が考えられる。

|      |              |
|------|--------------|
| PUSH | BP           |
| MOV  | BP,SP        |
| MOV  | SI,[BP + 4]  |
| MOV  | DS,[BP + 6]  |
| MOV  | DI,[BP + 8]  |
| MOV  | CX,[BP + 10] |

図4-7 スタックとインダイレクト・アドレッシングによる移動レジスタ初期設定

これらの命令は、必要な初期設定を行なう。そして、ルーチンは次の命令で終了する。

```
MOV    SP,BP
POP    BP
RET    8
```

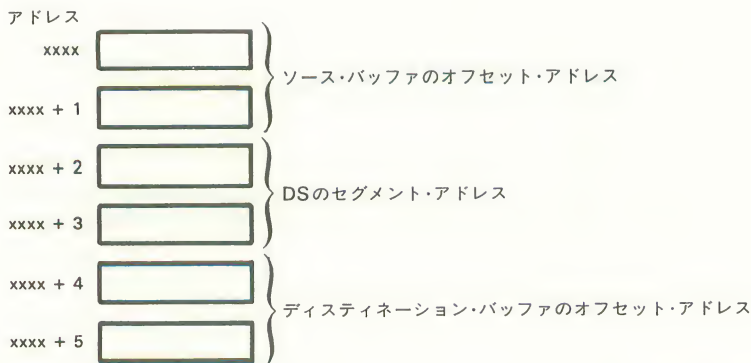
これは、リターン・アドレスをプログラム・カウンタに移動して、調整されたスタック・ポインタに8を加える。こうして、呼び出し元ルーチンでプッシュされたパラメータを、スタックから取り除く。

もしバッファがカレント・データ・セグメント内に位置すれば、バッファ・アドレスはLEA (Load Effective Address) 命令を用いてロードできる。次の命令は、LEA命令を用いてSIとDIにロードを行ない、MOV命令を用いてCXにメモリからCOUNTのデータをロードしている。

|     |              |
|-----|--------------|
| LEA | SI,BUFFER\$A |
| LEA | DI,BUFFER\$B |
| MOV | CX,COUNT     |

図4-8 LEA命令を用いたバッファ移動レジスタ初期設定

他の例として、BUFFER\$Aの最初の2バイトがバッファ内のバイト数、したがって移動されるバイト数を含むと仮定する。このとき、パラメータ・ブロックは次のようになる。



もし、DIレジスタがこのパラメータ・ブロックを示すとすれば、次の初期設定が用いられる。

```

LDS    SI, [DI]
MOV     DI, [DI+4]
MOV     CX, [SI]
INC     SI
INC     SI

```

2つのバッファ初期設定ルーチンを次に示す。第1のルーチンは8ビットのパターンをバッファ全体に写し、第2のルーチンは16ビットのパターンをバッファ全体に写す。このようなルーチンはバッファをクリヤするのによく用いられ、この場合、8ビットあるいは16ビットの値は0である。バイト長が奇数の短いバッファをクリヤするには最初のルーチンを用い、偶数バイト長のバッファあるいは奇数バイト長の長いバッファ（奇数バイトのクリヤに1バイト処理命令を用いる）には第2のルーチンを用いる。バッファを0でないあるパターンで初期設定する必要が生じる場合がある。たとえば、結果的にASCIIキャラクタのストリングを保持するバッファの初期設定には、ASCIIのスペース・コードが用いられる。

図4-9と図4-10に示すルーチンは、DIレジスタがディスティネーション・バッファを示していることを仮定している。ALあるいはAXのレジスタは、バッファ全体に写される8ビットあるいは16ビットの値を含む。CXレジスタは、バッファ内のバイトあるいはワードの数を指定する。

## (2) バッファの初期設定

バッファ初期設定ルーチンは、任意のデータをメモリのバッファにロードする。

|                   |     |                  |                          |
|-------------------|-----|------------------|--------------------------|
| INITIALIZE\$LOOP: | MOV | [DI],AL          | ;STORE INITIALIZING DATA |
|                   | INC | DI               | ;ADJUST POINTER          |
|                   | DEC | CX               | ;DECREMENT AND BRANCH    |
|                   | JNZ | INITIALIZE\$LOOP | ;if not done             |
|                   | RET |                  |                          |

図 4-9 バッファ初期設定 (8ビット・データ要素)

|                   |     |                  |                          |
|-------------------|-----|------------------|--------------------------|
| INITIALIZE\$LOOP: | MOV | [DI],AX          | ;STORE INITIALIZING DATA |
|                   | INC | DI               |                          |
|                   | INC | DI               |                          |
|                   | DEC | CX               |                          |
|                   | JNZ | INITIALIZE\$LOOP |                          |
|                   | RET |                  |                          |

図4-10 バッファ初期設定 (16ビット・データ要素)

2つのバッファ初期設定プログラムで、BXあるいはSIのレジスタをDIの代わりに用いることができる。

しばしば、バッファの最初のnバイトはバッファの記述に用いられる。たとえば、バッファ全体の長さとし頭の空のバイトに対するディスプレイメントは、バッファの最初の2バイトにストアされる。このバッファ記述用のバイトは、データがバッファに書かれるときは、調整される必要がある。

バッファ初期設定ルーチンは、バッファからバッファへの移動ルーチンで述べたと同じく、それ自体レジスタを初期設定する必要がある。

一般に、アドレスやカウンタの情報は、以下のどれかの方法でルーチンに渡される。

- パラメータ・ブロック
- スタック
- イミディエイト・データ
- LEA命令で用いられるアドレス

### (3) バッファの変換

バッファが変換されるとき、バッファ内のすべての要素は変換用のテーブルを用いて変換される。変換テーブルは、要素が持つすべての初期値に対して、直接に置換する値を与える。たとえば、バッファが1バイト要素から成るならば、各要素が持つことのできる、256個の可能性のある初期値が存在し、同様に同じ要素を持つことのできる256個の変換値が存在する。変換テーブルは、各初期値と変換値との結合を行なう。おそらく最も多く見られる変換テーブルは、ASCIIとEBCDICのキャラクタ間の変換で、各々はバイト値として符号化されている。この場合、ASCIIキャラクタのバッファが変換されれば、結果は等価なEBCDICキャラクタのバッファになる。

バッファが変換される次の2つの方法を考える。

1. バッファ内のデータが変換されてそのバッファに残る。
2. データが変換されて1つのバッファから他へ移動する。

図4-11のルーチンは、データを移動しないで変換する。このルーチンは、BXレジスタ

が変換テーブルのアドレスを含み、S Iレジスタが変換されるべきバッファのアドレスを含み、C Xレジスタが変換されるべきデータ要素の数を含むと仮定している。

|                  |      |                 |                                   |
|------------------|------|-----------------|-----------------------------------|
| TRANSLATE\$LOOP: | MOV  | AL,[SI]         | ;LOAD FROM BUFFER                 |
|                  | XLAT |                 | ;INDEX INTO TABLE                 |
|                  | MOV  | [SI],AL         | ;STORE CONVERTED DATA INTO BUFFER |
|                  | INC  | SI              | ;POINT AT NEXT ELEMENT            |
|                  | DEC  | CX              | ;DECREMENT AND TEST FOR DONE      |
|                  | JNZ  | TRANSLATE\$LOOP |                                   |
|                  | RET  |                 |                                   |

図4-11 バッファ内容の変換

図4-11のルーチンは、変換される要素が256バイト・テーブルに写されることを仮定している。この仮定で、X L A T命令を用いることが可能となる。変換されるべきデータ要素が16ビットのデータ単位であれば、より大きいテーブルが必要となる。図4-12のルーチンは、16ビットのデータ要素を65キロバイト\* のテーブルに写し、8ビットの結果を得る。

|                  |     |                 |                       |
|------------------|-----|-----------------|-----------------------|
| TRANSLATE\$LOOP: | MOV | DI,[SI]         | ;LOAD ELEMENT         |
|                  | MOV | AX,[BX + DI]    | ;USE ELEMENT AS INDEX |
|                  | MOV | [SI], AX        | ;STORE RESULT         |
|                  | INC | SI              | ;UPDATE POINTERS      |
|                  | INC | SI              |                       |
|                  | DEC | CX              | ;DECREMENT AND TEST   |
|                  | JNZ | TRANSLATE\$LOOP | ;FOR DONE             |
|                  | RET |                 |                       |

図4-12 16ビット・データ要素の変換

|                  |      |                 |                                             |
|------------------|------|-----------------|---------------------------------------------|
| TRANSLATE\$LOOP: | MOV  | AL,[SI]         | ;LOAD ELEMENT FROM SOURCE BUFFER            |
|                  | XLAT |                 | ;TRANSLATE DATA                             |
|                  | MOV  | [DI],AL         | ;STORE CONVERTED DATA IN DESTINATION BUFFER |
|                  | INC  | SI              | ;UPDATE POINTERS                            |
|                  | INC  | DI              |                                             |
|                  | DEC  | CX              | ;DECREMENT AND TEST FOR DONE                |
|                  | JNZ  | TRANSLATE\$LOOP |                                             |
|                  | RET  |                 |                                             |

図4-13 バッファからバッファへの変換移動

図4-13のルーチンは、データを変換して、1つのバッファから他へ移動する。このルーチンは、B Xレジスタが変換テーブルのアドレスを含み、S Iレジスタが変換されるバッファのアドレスを含み、D Iレジスタが変換されたデータのストアされるバッファのアドレスを含み、C Xレジスタが変換されるデータ要素の数を含むと仮定している。

図4-13のルーチンは、両方のバッファがD Sレジスタで示されるセグメント内に存在す

\*  $2^{16} = 65,536$  (訳者注)

ることを仮定している。

多くの変換ルーチンでは、変換バッファ内のすべての要素が特定の境界値内にあるかのチェックも行なう。図4-11から図4-13までのルーチンは、そのようなチェックを含むように、この章の後で更新を行なう。

前記のルーチンのレジスタ初期設定は、バッファからバッファへの移動ルーチンに用いられている初期設定の方法と類似している。

#### 4.1.2 CPUの状態の退避

8086は14個の16ビット・レジスタを持つ。ほとんどの場合、多くのレジスタを持つことは非常に望ましい。しかし、適当な処理が行なわれる間、CPU全体の状態がセーブされる必要があり、次いでCPUの状態が復元される必要のあるときに、多数のレジスタはあまり価値がない。

たとえば、ハードウェアあるいはソフトウェアの割り込みが発生すると、インタラプト・アクノリッジ・シーケンスで、8086は、フラグ・レジスタ、プログラム・カウンタ、コード・セグメント・レジスタの内容をセーブする。インタラプト・サービス・ルーチンでは、CPUの完全な状態をセーブしなければならない。次に示す一連の命令は、この作業を行なう。

レジスタをプッシュすべき特定の順序は存在しない。ただし、レジスタはそれがプッシュされた逆の順序で復元される必要がある。もしレジスタが図4-14に示す方法でセーブされれば、次に示されている方法がレジスタを復元するために用いられなければならない。

```
PUSH ES
PUSH DS
PUSH SI
PUSH DI
PUSH BP
PUSH DX
PUSH CX
PUSH BX
PUSH AX
```

図4-14 8086レジスタのセーブ

```
POP AX
POP BX
POP CX
POP DX
POP BP
POP DI
POP SI
POP DS
POP ES
```

図4-15 8086レジスタの回復



CPU全体の状態をセーブするためには、11バイトの命令コードと110クロックの期間を要する。この110サイクルには、8086が割り込みに応答するための時間は含まれていない。ハードウェア・インタラプトに対して、8086は割り込みを受け付けるために62クロックの期間を要する。ソフトウェア・インタラプトに対して、8086は割り込みを受け付けるために51から53クロックの期間を要する。この応答のためのサイクルは、割り込みが発生した間の命令実行に続いている。

したがって、CPU全体の状態の退避は172クロックの期間をも要し、これは5MHzの8086では32.4マイクロ秒になる。CUPの状態の復元には110クロックの期間を要し、さらにIRET (Interrupt Return) 命令の24クロックが加わる。以上から8086のCPUの状態を退避して復元するためには、1つの割り込みについて、306クロックの期間あるいは61.2マイクロ秒を要する。

#### 4.1.3 セグメント・レジスタの初期設定

プログラムがオペレーティング・システムと共に動作するように書かれていれば、オペレーティング・システムは一般にセグメント・レジスタの初期設定を行ない、続いて必要に応じてその内容を変更する。プログラムがオペレーティング・システムの恩恵を受けずに動作するように書かれていれば、セグメント・レジスタの初期設定を行なう必要がある。図4-16に示す命令は、セグメント・レジスタの初期設定を行なう。

|     |                        |                              |
|-----|------------------------|------------------------------|
| MOV | AX, IMM\$DATA\$FOR\$DS | ;LOAD IMMEDIATE DATA INTO AX |
| MOV | DS, AX                 |                              |
| MOV | AX, IMM\$DATA\$FOR\$ES | ;LOAD IMMEDIATE DATA INTO AX |
| MOV | ES, AX                 |                              |
| MOV | AX, IMM\$DATA\$FOR\$SS | ;LOAD IMMEDIATE DATA INTO AX |
| MOV | SS, AX                 |                              |

図4-16 イミディエイト・データによるESレジスタの初期設定

セグメント・レジスタの初期設定を行なうもう1つの方法は、図4-17に示すように、メモリからセグメント・レジスタにデータを直接移動すればよい。

|     |                       |
|-----|-----------------------|
| MOV | DS, CS: DATA\$FOR\$DS |
| MOV | ES, CS: DATA\$FOR\$ES |
| MOV | SS, CS: DATA\$FOR\$SS |

図4-17 コード・セグメント指定によるESレジスタの初期設定

セグメント・レジスタのESとSSに対するデータがDSで示されるセグメントに含まれていれば、第2と第3の命令のセグメント・プレフィックスは省略できる。

8086は、特定の初期設定の命令について、特殊な保護を行なう。SSとSPのレジスタが連続したMOV命令で初期設定されるとき、8086はこのMOV命令の間で割り込みが発生するのを禁止する。したがって、

```
MOV    SS, CS: DATA$FOR$SS
MOV    SP, DATA$FOR$SP
```



の一連の命令では、割り込みは生じない。

## 4.2 算術演算命令

8086の算術演算命令には、次の5つのタイプがある。

1. 加算命令
2. 減算命令
3. 乗算命令
4. 除算命令
5. 比較命令

上記分類の各々は、比較命令を除いて、ASCII/BCDの操作に用いることができる。

### 4.2.1 加算命令

種々のタイプの加算を行なう命令を表4-2に示す。図4-18、4-19、4-20は、いろいろな加算命令の使用を示している。各ルーチンは、加算される数あるいはストリングはデータ・セグメントに存在し、次の順序になっていると仮定している。



#### (1) 1組の複数ワード数値の和

図4-18のルーチンは、SIとDIのレジスタが加算される複数ワード数値の先頭アドレスを含み、CXレジスタが加算されるワード数を含むと仮定している。結果は、DIレジスタで示されるストリングにストアされる。

|                 |     |                |                                   |
|-----------------|-----|----------------|-----------------------------------|
| START:          | CLC |                | ;CLEAR CARRY FOR INITIAL ADDITION |
| ADDITION\$LOOP: | MOV | AX,[SI]        | ;LOAD FROM INITIAL STRING         |
|                 | ADC | [DI],AX        | ;ADD AX TO MEMORY                 |
|                 | INC | SI             | ;UPDATE POINTERS                  |
|                 | INC | SI             |                                   |
|                 | INC | DI             |                                   |
|                 | INC | DI             |                                   |
|                 | DEC | CX             |                                   |
|                 | JNZ | ADDITION\$LOOP |                                   |
|                 | RET |                |                                   |

図4-18 複数ワード加算

ストリング・プリミティブとLOOP命令によって、メモリの数とこのルーチン実行に必要な時間が減少できる。

#### (2) 1組の複数バイトBCD数値の和

図4-19のルーチンは、SIとDIのレジスタが加算されるBCDストリングの先頭アドレスを含み、CXレジスタが各ストリングのBCDバイト数を含むと仮定している。結果はDIレジスタで示されるストリングにストアされる。

表 4-2 8086の加算命令

| ニーモニック | オペランド             | オブジェクト・コード                                                          | バイト           | クロック                                                    | ステータス |   |   |   |   |   |   |   | 動作内容                                                                                                                                                                                                                |
|--------|-------------------|---------------------------------------------------------------------|---------------|---------------------------------------------------------|-------|---|---|---|---|---|---|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                   |                                                                     |               |                                                         | O     | D | I | T | S | Z | A | P | C                                                                                                                                                                                                                   |
| ADC    | mem/reg, mem/reg2 | 000100dw<br>mod rrr/r/m<br>(DISP)<br>(DISP)                         | 2, 3 または 4    | reg - reg: 3<br>mem - reg: 9 + EA<br>reg - mem: 16 + EA | X     |   |   | X | X | X | X | X | $(\text{mem}/\text{reg}) \leftarrow (\text{mem}/\text{reg}) + (\text{mem}/\text{reg2}) + [\text{CF}]$<br>mem/reg2 で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容とキャリー・フラグを、mem/reg で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容に加算する。 |
| ADC    | mem/reg, data     | 100000sw<br>mod 010 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (sw=01 のとき) | 3, 4, 5 または 6 | reg: 4<br>mem: 17 + EA                                  | X     |   |   | X | X | X | X | X | $(\text{mem}/\text{reg}) \leftarrow (\text{mem}/\text{reg}) + \text{data} + [\text{CF}]$<br>8 あるいは 16 ビットのイミディエイト・データとキャリー・フラグを、mem/reg で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容に加算する。                                  |
| ADC    | ac, data          | 0001010w<br>kk<br>jj (w=1 のとき)                                      | 2 または 3       | 4                                                       | X     |   |   | X | X | X | X | X | $(\text{ac}) \leftarrow (\text{ac}) + \text{data} + [\text{CF}]$<br>8 あるいは 16 ビットのイミディエイト・データとキャリー・フラグを、AL (8 ビット操作) あるいは AX (16 ビット操作) のレジスタに加算する。                                                                 |
| ADD    | mem/reg, mem/reg2 | 000000dw<br>mod rrr/r/m<br>(DISP)<br>(DISP)                         | 2, 3 または 4    | reg - reg: 3<br>mem - reg: 9 + EA<br>reg - mem: 16 + EA | X     |   |   | X | X | X | X | X | $(\text{mem}/\text{reg}) \leftarrow (\text{mem}/\text{reg}) + (\text{mem}/\text{reg2})$<br>mem/reg2 で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容を、mem/reg1 で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容に加算する。                       |
| ADD    | mem/reg, data     | 100000sw<br>mod 000 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (sw=01 のとき) | 3, 4, 5 または 6 | reg: 4<br>mem: 17 + EA                                  | X     |   |   | X | X | X | X | X | $(\text{mem}/\text{reg}) \leftarrow (\text{mem}/\text{reg}) + \text{data}$<br>8 あるいは 16 ビットのイミディエイト・データを、mem/reg で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容に加算する。                                                         |
| ADD    | ac, data          | 0000010w<br>kk<br>jj (w=1 のとき)                                      | 2 または 3       | 4                                                       | X     |   |   | X | X | X | X | X | $(\text{ac}) \leftarrow (\text{ac}) + \text{data}$<br>8 あるいは 16 ビットのイミディエイト・データを、AL (8 ビット操作) あるいは AX (16 ビット操作) のレジスタに加算する。                                                                                        |
| INC    | mem/reg           | 1111111w<br>mod 000 r/m<br>(DISP)<br>(DISP)                         | 2, 3 または 4    | reg: 3<br>mem: 15 + EA                                  | X     |   |   | X | X | X | X | X | $(\text{mem}/\text{reg}) \leftarrow (\text{mem}/\text{reg}) + 1$<br>mem/reg で示されるメモリ位置あるいはレジスタの 8 あるいは 16 ビットの内容を、1 だけ増やす。                                                                                          |

表 4-2 8086の加算命令 (続き)

| ニーモニック | オペランド | オブジェクト・コード | バイト | クロック | ステータス |   |   |   |   |   |   |   |   |                                                  | 動作内容 |
|--------|-------|------------|-----|------|-------|---|---|---|---|---|---|---|---|--------------------------------------------------|------|
|        |       |            |     |      | O     | D | I | T | S | Z | A | P | C |                                                  |      |
| INC    | reg   | 01000rrr   | 1   | 2    | X     |   |   |   | X | X | X | X |   | (reg) ← (reg) + 1<br>指定されたレジスタの16ビットの内容を、1だけ増やす。 |      |
| AAA    |       | 37         | 1   | 4    | ?     |   |   |   | ? | ? | X | ? | X | ASCIIによる加算後にALレジスタの内容をアジャストする。                   |      |
| DAA    |       | 27         | 1   | 4    | ?     |   |   |   | X | X | X | X | X | 10進による加算後にALレジスタの内容をアジャストする。                     |      |

|                      |     |                     |                                   |
|----------------------|-----|---------------------|-----------------------------------|
| START:               | CLC |                     | ;CLEAR CARRY FOR INITIAL ADDITION |
| BCD\$ADDITION\$LOOP: | MOV | AL,[SI]             | ;LOAD FROM STRING A               |
|                      | ADC | AL,[DI]             | ;ADD FROM STRING B                |
|                      | DAA |                     | ;PERFORM BCD ADJUST               |
|                      | MOV | [DI],AL             | ;STORE RESULT                     |
|                      | INC | SI                  | ;UPDATE POINTERS                  |
|                      | INC | DI                  |                                   |
|                      | DEC | CX                  | ;DECREMENT AND TEST               |
|                      | JNZ | BCD\$ADDITION\$LOOP | ;FOR DONE                         |
|                      | RET |                     |                                   |

図4-19 複数バイトのBCD加算

### (3) 1組の複数バイトASCIIストリングの和

図4-20のルーチンは、SIとDIのレジスタが加算される2つのASCIIストリングの先頭アドレスを含むと仮定している。CXレジスタは、各ストリングのASCIIバイト数を含む。結果は、DIレジスタで示されるストリングにストアされる。

|                        |     |                       |                     |
|------------------------|-----|-----------------------|---------------------|
| ASCII\$ADDITION\$LOOP: | CLC |                       |                     |
|                        | MOV | AL,[SI]               | ;LOAD FROM STRING A |
|                        | ADC | AL,[DI]               | ;ADD STRING B       |
|                        | AAA |                       | ;PERFORM AN ADJUST  |
|                        | MOV | [DI],AL               | ;STORE RESULT       |
|                        | INC | SI                    | ;ADJUST POINTERS    |
|                        | INC | DI                    |                     |
|                        | DEC | CX                    | ;DECREMENT AND TEST |
|                        | JNZ | ASCII\$ADDITION\$LOOP | ;FOR DONE           |
|                        | RET |                       |                     |

図4-20 複数バイトのASCII加算

図4-19と図4-20のルーチンは共に、操作を行なうのに必要なバイト数と時間を減らすために、ストリング・プリミティブを用いることができる。

前記の加算ルーチンに対して、加算される数が次の形式である場合を考える。

バイト#0  オペランドの最上位バイト

バイト#n  オペランドの最下位バイト

この場合、加算ルーチンは図4-18から図4-20のものとは、次の2つの主要な点で異なる。

1. 初期設定が異なる。初期設定では、レジスタは複数バイト数値の最初ではなく最後を示している。
2. ポインタは、増加ではなく、減少する。

この差を補償するためには、変更された先頭アドレスを適当なアドレス・レジスタにロードする必要がある。続いて、アドレスは減少させなければならない。

#### 4.2.2 減算命令

減算命令を表4-3に示す。前節の複数バイトの加算ルーチンを減算に直したものは容易に得られる。このルーチンの作成は、読者に練習問題として残す。

表 4-3 8086の減算命令

| ニーモニック | オペランド                | オブジェクト・コード                                                         | バイト             | クロック                                                          | ステータス |   |   |   |   |   |   |   | 動作内容 |                                                                                                                                                           |
|--------|----------------------|--------------------------------------------------------------------|-----------------|---------------------------------------------------------------|-------|---|---|---|---|---|---|---|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                      |                                                                    |                 |                                                               | O     | D | I | T | S | Z | A | P |      | C                                                                                                                                                         |
| SUB    | mem/reg,<br>mem/reg2 | 001010dw<br>mod rrr r/m<br>(DISP)<br>(DISP)                        | 2, 3ま<br>たは4    | reg - reg: 3<br>mem - reg:<br>9 + EA<br>reg - mem:<br>16 + EA | X     |   |   |   | X | X | X | X | X    | (mem/reg) ← (mem/reg) - (mem/reg2)<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容から、mem/reg2で示されるメモリ<br>位置あるいはレジスタの8あるいは16ビットの内容を減<br>じる。                 |
| SUB    | mem/reg,<br>data     | 100000sw<br>mod 101 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (sw=01のとき) | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>17 + EA                                     | X     |   |   |   | X | X | X | X | X    | (mem/reg) ← (mem/reg) - data<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容から、8あるいは16ビットのイミ<br>ディエイト・データを減じる。                                             |
| SUB    | ac,data              | 0010110w<br>kk<br>jj (w=10のとき)                                     | 2ま<br>たは3       | 4                                                             | X     |   |   |   | X | X | X | X | X    | (ac) ← (ac) - data<br>AL (8ビット操作) あるいはAX (16ビット操作) のレ<br>ジスタから、8あるいは16ビットのイミディエイト・デ<br>ータを減じる。                                                            |
| SBB    | mem/reg,<br>mem/reg2 | 000110dw<br>mod rrr r/m<br>(DISP)<br>(DISP)                        | 2, 3ま<br>たは4    | reg - reg: 3<br>mem - reg:<br>9 + EA<br>reg - mem:<br>16 + EA | X     |   |   |   | X | X | X | X | X    | (mem/reg) ← (mem/reg) - (mem/reg2) - (CF)<br>mem/regで示されるメモリ位置あるいはレジスタの8<br>あるいは16ビットの内容から、mem/reg2で示されるメ<br>モリ位置あるいはレジスタの8あるいは16ビットの内<br>容とキャリー・フラグを減じる。 |
| SBB    | mem/reg,<br>data     | 100000sw<br>mod 011 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (sw=01のとき) | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>17 + EA                                     | X     |   |   |   | X | X | X | X | X    | (mem/reg) ← (mem/reg) - data - (CF)<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容から、8あるいは16ビットのイミ<br>ディエイト・データとキャリー・フラグを減じる。                             |
| SBB    | ac,data              | 0001110w<br>kk<br>jj (w=10のとき)                                     | 2ま<br>たは3       | 4                                                             | X     |   |   |   | X | X | X | X | X    | (ac) ← (ac) - data - (CF)<br>AL (8ビット操作) あるいはAX (16ビット操作) のレ<br>ジスタから、8あるいは16ビットのイミディエイト・デ<br>ータとキャリー・フラグを減じる。                                            |
| DEC    | mem/reg              | 1111111w<br>mod 001 r/m<br>(DISP)<br>(DISP)                        | 2, 3ま<br>たは4    | reg: 3<br>mem:<br>15 + EA                                     | X     |   |   |   | X | X | X | X | X    | (mem/reg) ← (mem/reg) - 1<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容を1だけ減じる。                                                                          |



表 4-3 8086の減算命令 (続き)

| ニーモニック | オペランド   | オブジェクトコード                                 | バイト        | クロック                      | ステータス |   |   |   |   |   |   |   | 動作 内 容 |                                                                                                                |
|--------|---------|-------------------------------------------|------------|---------------------------|-------|---|---|---|---|---|---|---|--------|----------------------------------------------------------------------------------------------------------------|
|        |         |                                           |            |                           | O     | D | I | T | S | Z | A | P |        | C                                                                                                              |
| DEC    | reg     | 01001rr                                   | 1          | 2                         | X     |   |   |   | X | X | X | X | X      | $[\text{reg}] \leftarrow [\text{reg}] - 1$<br>指定されたレジスタの16ビットの内容を1だけ減じる。                                       |
| AAS    |         | 3F                                        | 1          | 4                         | ?     |   |   |   | ? | ? | X | ? | X      | ASCII による減算後の AL レジスタの内容をアジャストする。                                                                              |
| DAS    |         | 2F                                        | 1          | 4                         | ?     |   |   |   | X | X | X | X | X      | 10進による減算後の AL レジスタの内容をアジャストする。                                                                                 |
| NEG    | mem/reg | 1111011w<br>mod011r/m<br>(DISP)<br>(DISP) | 2, 3 または 4 | reg: 3<br>mem:<br>16 + EA | X     |   |   |   | X | X | X | X | X      | $[\text{reg}] \leftarrow \overline{[\text{reg}] + 1}$<br>mem/regで示されるメモリ位置あるいはレジスタの、8<br>あるいは16ビットの内容の2の補数をとる。 |



### 4.2.3 乗算命令

種々のタイプの乗算を行なう8086の命令を表4-4に示す。

図4-21と図4-22のルーチンは、8086の乗算命令の代表的な使用を示している。

#### (1) 32ビットと32ビットの乗算

図4-21のルーチンは、2つの32ビット符号なし数値を乗じて、64ビットの結果を与える。このルーチンは次の形式のデータ・ブロックに作用する。

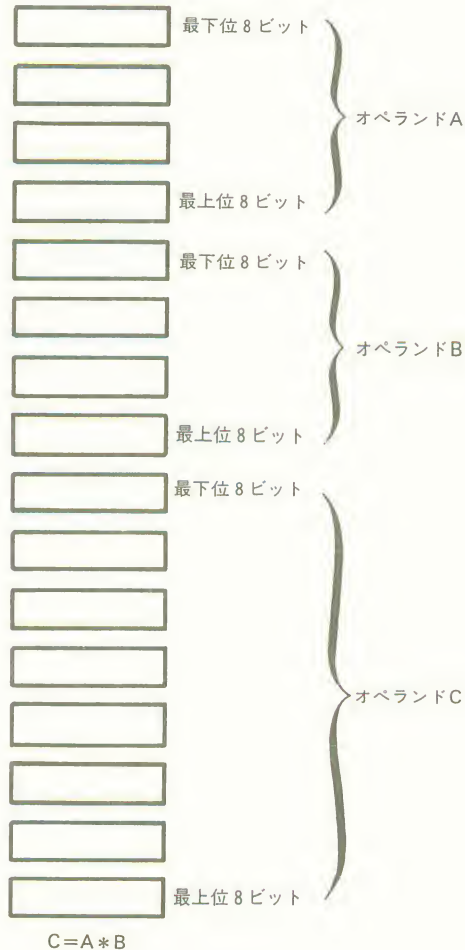


図4-21のルーチンは、BXレジスタがこのブロックを示していることを仮定している。

このような処理に対するもう1つの方法は、8086システムへの8087ニューメリック・コープロセッサの付加である。8087は、単精度と倍精度の浮動小数点演算と同様に、32ビットと64ビットの整数演算を行なう。

表 4-4 8086の乗算命令

| ニーモニック | オペランド           | オブジェクトコード                         | バイト      | クロック                | ステータス |   |   |   |   |   |   |   |   |  | 動作内容                                                                                                                                                                                                                                                |
|--------|-----------------|-----------------------------------|----------|---------------------|-------|---|---|---|---|---|---|---|---|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                 |                                   |          |                     | O     | D | I | T | S | Z | A | P | C |  |                                                                                                                                                                                                                                                     |
| MUL    | reg<br>(8-bit)  | 11110110                          | 2        | 70 → 77             | X     |   |   |   |   |   |   |   |   |  | w=0のとき、[AX]←[AL]・[mem/reg]<br>w=1のとき、[DX]←[AX]・[mem/reg]<br>mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容と、AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を乗じる。結果は、8・8ビット操作の場合、AXレジスタにストアされ、16・16ビット操作の場合、DXレジスタ (上位16ビット) とAXレジスタ (下位16ビット) にストアされる。内容は符号なしの乗算である。 |
|        | reg<br>(16-bit) | 11110111                          | 2        | 118 → 133           |       |   |   |   |   |   |   |   |   |  | 実行時間は、8ビット操作で7クロック、16ビット操作で15クロックの変動がある。                                                                                                                                                                                                            |
|        | mem<br>(8-bit)  | 11110110<br>mod 100 r m<br>(DISP) | 2, 3または4 | (76 → 83)<br>+ EA   |       |   |   |   |   |   |   |   |   |  | 号なしの乗算である。                                                                                                                                                                                                                                          |
|        | mem<br>(16-bit) | 11110111<br>mod 100 r m<br>(DISP) | 2, 3または4 | (124 → 139)<br>+ EA |       |   |   |   |   |   |   |   |   |  | 実行時間は、8ビット操作で7クロック、16ビット操作で15クロックの変動がある。                                                                                                                                                                                                            |
| IMUL   | reg<br>(8-bit)  | 11110110                          | 2        | 80 → 98             | X     |   |   |   |   |   |   |   |   |  | w=0のとき、[AX]←[AL]・[mem/reg]<br>w=1のとき、[DX]←[AX]・[mem/reg]<br>mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容と、AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を乗じる。結果は、8・8ビット操作の場合、AXレジスタにストアされ、16・16ビット操作の場合、DXレジスタ (上位16ビット) とAXレジスタ (下位16ビット) にストアされる。内容は符号付き乗算である。  |
|        | reg<br>(16-bit) | 11110111                          | 2        | 128 → 154           |       |   |   |   |   |   |   |   |   |  | 実行時間は、8ビット操作で18クロック、16ビット操作で26クロックの変動がある。                                                                                                                                                                                                           |
|        | mem<br>(8-bit)  | 11110110<br>mod 101 r m<br>(DISP) | 2, 3または4 | (86 → 104)<br>+ EA  |       |   |   |   |   |   |   |   |   |  | 号付き乗算である。                                                                                                                                                                                                                                           |
|        | mem<br>(16-bit) | 11110111<br>mod 101 r m<br>(DISP) | 2, 3または4 | (134 → 160)<br>+ EA |       |   |   |   |   |   |   |   |   |  | 実行時間は、8ビット操作で18クロック、16ビット操作で26クロックの変動がある。                                                                                                                                                                                                           |
| AAM    |                 | D4<br>0A                          | 2        | 83                  | ?     |   |   |   | X | X | ? | X | ? |  | 2つのアンパック形式の10進オペランドの乗算後に、アンパック形式の10進の結果を得るためにAXの値をアジャストする。                                                                                                                                                                                          |

w=0のとき,  $[AX] \leftarrow [AL] \cdot [mem/reg]$   
w=1のとき,  $[DX] : [AX] \leftarrow [AX] \cdot [mem/reg]$   
mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容と, AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を乗じる。結果は、 $8 \times 8$ ビット操作の場合、AXレジスタにストアされ、 $16 \times 16$ ビット操作の場合、DXレジスタ (上位16ビット) とAXレジスタ (下位16ビット) にストアされる。内容は符号なしの乗算である。  
実行時間は、8ビット操作で7クロック、16ビット操作で15クロックの変動がある。

w=0のとき,  $[AX] \leftarrow [AL] \cdot [mem/reg]$   
w=1のとき,  $[DX] : [AX] \leftarrow [AX] \cdot [mem/reg]$   
mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容と, AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を乗じる。結果は、 $8 \times 8$ ビット操作の場合、AXレジスタにストアされ、 $16 \times 16$ ビット操作の場合、DXレジスタ (上位16ビット) とAXレジスタ (下位16ビット) にストアされる。内容は符号付き乗算である。  
実行時間は、8ビット操作で18クロック、16ビット操作で26クロックの変動がある。

2つのアンパック形式の10進オペランドの乗算後に、アンパック形式の10進の結果を得るためにAXの積をアジャストする。



バイト#0  BCDによる最下位の桁

#1

・                   ・                   ・  
・                   ・                   ・  
・                   ・                   ・  
・                   ・                   ・

#n+1  BCDによる最上位の桁

|                       |     |                      |                                     |
|-----------------------|-----|----------------------|-------------------------------------|
|                       | MOV | [DI], 0              | ;CLEAR INITIAL BYTE OF BCD STRING   |
|                       | AND | DL,0FH               | ;AND OFF BITS 4 AND 5 OF MULTIPLIER |
| MULTIPLY\$NEXT\$BYTE: | MOV | AL,[SI]              | ;LOAD MULTIPLICAND                  |
|                       | INC | SI                   |                                     |
|                       | AND | AL,0FH               | ;CLEAR UPPER NIBBLE                 |
|                       | MUL | DL                   | ;MULTIPLY BCD * BCD                 |
|                       | AAM |                      | ;ADJUST RESULT                      |
|                       | ADD | AL,[DI]              | ;ADD IN BCD                         |
|                       | AAA |                      |                                     |
|                       | MOV | [DI],AL              | ;STORE RESULT                       |
|                       | INC | DI                   |                                     |
|                       | MOV | [DI],AH              |                                     |
|                       | DEC | CX                   | ;DECREMENT AND TEST FOR DONE        |
|                       | JNZ | MULTIPLY\$NEXT\$BYTE |                                     |
|                       | RET |                      |                                     |

図4-22 ASCIIによる乗算

4.2.4 除算命令

種々の除算処理を行なう8086の命令を表4-5に示す。

図4-23のルーチンは、8086の除算命令の使用を示している。

(1) ASCIIによる除算

図4-23のルーチンは、ASCIIストリングのASCII1桁による除算を行なう。結果はBCD桁のストリングである。ルーチンは、ASCIIストリングが次のように構成されていることを仮定している。

バイト#0  最上位バイト

#1

・                   ・                   ・  
・                   ・                   ・  
・                   ・                   ・  
・                   ・                   ・

#n  最下位バイト

表 4-5 8086の除算命令

| ニーモニック | オペランド           | オブジェクト・コード                        | バイト      | クロック              | ステータス |   |   |   |   |   |   |   |   |   |                                                                                                                                                                                                                  | 動作 内 容 |
|--------|-----------------|-----------------------------------|----------|-------------------|-------|---|---|---|---|---|---|---|---|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
|        |                 |                                   |          |                   | O     | D | I | T | S | Z | A | P | C |   |                                                                                                                                                                                                                  |        |
| DIV    | reg<br>(8-bit)  | 11110110                          | 2        | 80—90             | ?     |   |   |   |   | ? | ? | ? | ? | ? | w 0のとき、<br>AH: 余り ← AX // mem/reg<br>AL: 商                                                                                                                                                                       |        |
|        | reg<br>(16-bit) | 11110111                          | 2        | 144—162           |       |   |   |   |   |   |   |   |   |   | w 1のとき、<br>DX: 余り ← DX : AX // mem/reg<br>AX: 商                                                                                                                                                                  |        |
|        | mem<br>(8-bit)  | 11110110<br>mod 110 r m<br>(DISP) | 2, 3または4 | (86—96)<br>+ EA   |       |   |   |   |   |   |   |   |   |   | 16ビット操作の場合はAXレジスタを、あるいは32ビット操作の場合はDXレジスタ(上位16ビット)とAXレジスタ(下位16ビット)を、mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容で割る。16ビットを8ビットで割る場合、商はALに置かれ、余りはAHにストアされる。32ビットを16ビットで割る場合、商はAXレジスタに置かれ、余りはDXレジスタに置かれる。これは符号なし除算操作である。 |        |
|        | mem<br>(16-bit) | 11110111<br>mod 110 r m<br>(DISP) | 2, 3または4 | (150—168)<br>+ EA |       |   |   |   |   |   |   |   |   |   | 実行時間は、8ビット操作では10クロックの、16ビット操作では18クロックの変動がある。この変動はデータに依存する。                                                                                                                                                       |        |
| IDIV   | reg<br>(8-bit)  | 11110110                          | 2        | 101—112           | ?     |   |   |   |   | ? | ? | ? | ? | ? | w 0のとき、<br>AH: 余り ← AX // mem/reg<br>AL: 商                                                                                                                                                                       |        |
|        | reg<br>(16-bit) | 11110111                          | 2        | 165—184           |       |   |   |   |   |   |   |   |   |   | w 1のとき、<br>DX: 余り ← DX : AX // mem/reg<br>AX: 商                                                                                                                                                                  |        |
|        | mem<br>(8-bit)  | 11110110<br>mod 111 r m<br>(DISP) | 2, 3または4 | (107—118)<br>+ EA |       |   |   |   |   |   |   |   |   |   | 16ビット操作の場合はAXレジスタを、あるいは32ビット操作の場合はDXレジスタ(上位16ビット)とAXレジスタ(下位16ビット)を、mem/regで示されるメモリ位置あるいはレジスタの8あるいは16ビットの内容で割る。16ビットを8ビットで割る場合、商はALに置かれ、余りはAHにストアされる。32ビットを16ビットで割る場合、商はAXレジスタに置かれ、余りはDXレジスタに置かれる。これは符号付き除算操作である。 |        |
|        | mem<br>(16-bit) | 11110111<br>mod 111 r m<br>(DISP) | 2, 3または4 | (171—190)<br>+ EA |       |   |   |   |   |   |   |   |   |   | 実行時間は、8ビット操作では11クロックの、16ビット操作では19クロックの変動がある。この変動はデータに依存する。                                                                                                                                                       |        |

表 4-5 8086の除算命令 (続き)

| ニーモニック | オペランド | オブジェクト・コード | バイト | クロック | ステータス |   |   |   |   |   |   |   |   |                                                | 動作内容                                                            |
|--------|-------|------------|-----|------|-------|---|---|---|---|---|---|---|---|------------------------------------------------|-----------------------------------------------------------------|
|        |       |            |     |      | O     | D | I | T | S | Z | A | P | C |                                                |                                                                 |
| CBW    |       | 98         | 1   | 2    |       |   |   |   |   |   |   |   |   | [AH] ← [AL]<br>ALレジスタの符号ビット、ビット7をAHレジスタに拡張する。  |                                                                 |
| CWD    |       | 99         | 1   | 5    |       |   |   |   |   |   |   |   |   | [DX] ← [AX]<br>AXレジスタの符号ビット、ビット15をDXレジスタに拡張する。 |                                                                 |
| AAD    |       | D5<br>0A   | 2   | 60   | ?     |   |   |   | X | X | ? | ? | X | ?                                              | アンパック形式の10進の商を得るために、アンパック形式の10進の除数の除算に先立って、ALの被除数の10進アジャストを行なう。 |



S IレジスタはASCIIストリングを示し、D Lレジスタは除数であるASCIIの1桁を含み、D Iレジスタは結果のBCDストリングがストアされるメモリ位置を示し、C Xレジスタは被除数の桁数を含む。BCDストリングでストアされる結果は、次の形式となる。

```

      バイト#0    最上位バイト
                .
      #1  
                .
                .
                .
      #n    最下位バイト
  
```

|                     |     |                    |                              |
|---------------------|-----|--------------------|------------------------------|
|                     | AND | DL,0FH             | ;CLEAR HIGH-ORDER NIBBLE     |
|                     | XOR | AH,AH              | ;CLEAR AH                    |
| DIVIDE\$NEXT\$BYTE: | MOV | AL,[SI]            | ;LOAD BYTE FROM ASCII STRING |
|                     | INC | SI                 |                              |
|                     | AND | AL,0FH             | ;CLEAR BITS 4 AND 5          |
|                     | AAD |                    | ;ADJUST USING AH             |
|                     | DIV | DL                 |                              |
|                     | MOV | [DI],AL            | ;STORE RESULT                |
|                     | INC | DI                 |                              |
|                     | DEC | CX                 | ;DECREMENT AND TEST FOR DONE |
|                     | JNZ | DIVIDE\$NEXT\$BYTE |                              |
|                     | RET |                    |                              |

図4-23 ASCIIによる除算

## (2) 64ビットの除算

64ビットの被除数を32ビットの除数で割ることは、8086では容易なことではない。D I VとI D I Vの命令は、この機能を行なうときに特に有用ではない。64ビット数値を32ビットで割るためには、減算とシフトのアルゴリズムを用いる必要がある。このようなルーチンの作成は、重要な作業で、ここでの解説の範囲を越えている。

このような処理に対するもう1つの方法は、8086システムへの8087ニューメリック・コプロセッサの付加である。8087は、単精度と倍精度の浮動小数点演算と同様に、32ビットと64ビットの整数演算を行なう。

### 4.2.5 比較命令

8086の比較命令を表4-6に示す。これらは減算命令のように動作するが、レジスタあるいはメモリ位置に結果を戻さない。減算操作はステータス・フラグを設定するためにだけ用いられる。

比較命令の使用を図4-24から図4-27に示す。

表 4-6 8086の比較命令

| ニーモニック | オペランド                 | オブジェクト・コード                                                         | バイト             | クロック                                                   | ステータス |   |   |   |   |   |   |   | 動作 内 容 |   |                                                                                                                                                                                                   |
|--------|-----------------------|--------------------------------------------------------------------|-----------------|--------------------------------------------------------|-------|---|---|---|---|---|---|---|--------|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                       |                                                                    |                 |                                                        | O     | D | I | T | S | Z | A | P |        | C |                                                                                                                                                                                                   |
| CMP    | mem/reg1,<br>mem/reg2 | 001110dw<br>mod rrr/r/m<br>(DISP)<br>(DISP)                        | 2, 3ま<br>たは4    | reg-reg: 3<br>mem-reg:<br>9 + EA<br>reg-mem:<br>9 + EA | X     |   |   |   |   | X | X | X | X      | X | $[(\text{mem}/\text{reg1}) - (\text{mem}/\text{reg2})]$<br>mem/reg1で示されるメモリ位置あるいはレジスタの8<br>あるいは16ビットの内容から、mem/reg2で示されるメ<br>モリ位置あるいはレジスタの8あるいは16ビットの内容<br>の減算を行なう。結果はフラグを設定するために用い<br>られてから捨てられる。 |
| CMP    | mem/reg,<br>data      | 100000sw<br>mod 111 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (sw=01のとき) | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>10 + EA                              | X     |   |   |   |   | X | X | X | X      | X | $[(\text{mem}/\text{reg}) - \text{data}]$<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容から、8あるいは16ビットのイミ<br>ディエイト・データの減算を行なう。結果はフラグを設<br>定するために用いられてから捨てられる。                                      |
| CMP    | ac,data               | 001110w<br>kk<br>jj (w=1のとき)                                       | 2また<br>は3       | 4                                                      | X     |   |   |   |   | X | X | X | X      | X | $[\text{ac}] - \text{data}$<br>AL (8ビット操作) あるいはAX (16ビット操作) のレジ<br>スタから、8あるいは16ビットのイミディエイト・デー<br>タの減算を行なう。結果はフラグを設定するために用い<br>られてから捨てられる。                                                         |

2つのストリング・プリミティブ命令、CMP SとSCASも比較動作を行なう。これらの命令については、この章の後で他のプリミティブ命令と共に述べる。

### (1) ストリングの長さの計算

図4-24のルーチンは、ストリング中のキャラクタの数を決める。

このルーチンは、SIレジスタが調べられるストリングのアドレスを示し、AHがストリングの終わりを示すキャラクタを含むと仮定している。このルーチンが実行を終えたとき、DXレジスタはストリングの先頭と終了キャラクタの間のキャラクタ数を含んでいる。

|                       |     |                      |                            |
|-----------------------|-----|----------------------|----------------------------|
|                       | MOV | DX,0FFFFH            | ;INITIALIZE COUNT TO -1    |
| SCAN\$FOR\$DELIMITER: | INC | DX                   | ;INCREMENT COUNT           |
|                       | MOV | AL,[SI]              | ;LOAD BYTE FROM STRING     |
|                       | INC | SI                   | ;UPDATE POINTER            |
|                       | CMP | AH,AL                | ;COMPARE WITH TERMINATION  |
|                       | JNZ | SCAN\$FOR\$DELIMITER | ;BRANCH IF NOT TERMINATION |
|                       | RET |                      |                            |

図4-24 ストリングの長さの計算

ストリング・プリミティブ命令SCASは、このルーチンのメモリの減少と実行を速めるために用いることができる。SCAS命令は、この章の後で他のストリング・プリミティブ命令と共に述べる。

### (2) 系列中で最大の8ビット符号なし数値を求める

図4-25のルーチンは、8ビット符号なし数値の系列中で最大値を決定する。このルーチンは、SIレジスタが調べられる一連の数値のアドレスを示し、CXレジスタが調べられるバイト数を含むと仮定している。このルーチンの実行が終了したとき、AHは最大値を含み、DXは最大値の位置を示している。

|                   |     |                  |                              |
|-------------------|-----|------------------|------------------------------|
|                   | XOR | AH,AH            | ;INITIALIZE MAX. NUMBER      |
| SCAN\$NEXT\$BYTE: | MOV | AL,[SI]          | ;LOAD BYTE FROM SEQUENCE     |
|                   | CMP | AH,AL            | ;COMPARE WITH CURRENT MAX. # |
|                   | JAE | UPDATE\$PTR      |                              |
|                   | MOV | AH,AL            | ;SAVE NEW MAX. NUMBER        |
| UPDATE\$PTR:      | MOV | DX,SI            | ;SAVE LOCATION OF MAX. #     |
|                   | INC | SI               |                              |
|                   | DEC | CX               |                              |
|                   | JNZ | SCAN\$NEXT\$BYTE |                              |
|                   | RET |                  |                              |

図4-25 最大の8ビット数値を求める

図4-25と図4-26のルーチンは、ストリング・プリミティブ命令を用いることによって改善できる。

### (3) 系列中で最大の16ビット数値を求める

図4-26のルーチンは、16ビット符号付き数値の系列中で最大値を決定する。このルーチンは、SIレジスタが調べられる数値列のアドレスを示し、CXレジスタが調べられるワード数を含むと仮定している。

|              |     |             |                                   |
|--------------|-----|-------------|-----------------------------------|
| SCAN\$LOOP:  | MOV | BX,8000H    | ;INITIALIZE MAX. NUMBER           |
|              | MOV | AX,[SI]     | ;LOAD NUMBER FROM SEQUENCE        |
|              | CMP | BX,AX       | ;COMPARE WITH CURRENT MAX. NUMBER |
|              | JGE | UPDATE\$PTR |                                   |
|              | MOV | BX,AX       | ;SAVE NEW MAX. NUMBER             |
| UPDATE\$PTR: | MOV | DX,SI       | ;SAVE LOCATION OF MAX. NUMBER     |
|              | INC | SI          | ;UPDATE PRT.                      |
|              | INC | SI          |                                   |
|              | DEC | CX          | ;DECREMENT AND TEST FOR DONE      |
|              | JNZ | SCAN\$LOOP  |                                   |
|              | RET |             |                                   |

図4-26 最大の16ビット数値を求める

#### (4) バッファの変換

この章の初めに、2つのバッファ変換ルーチンを示した。次のルーチンはエラーのチェックを含んでいる。変換されるバッファ内のキャラクタは、 $20_{16}$ から $5F_{16}$ までの範囲になければならない。このルーチンは、BXレジスタが変換テーブルのアドレスを含み、SIレジスタが変換されるバッファのアドレスを含み、CXレジスタが変換されるデータ要素の数を含むと仮定している。

|                   |      |                  |                               |
|-------------------|------|------------------|-------------------------------|
| TRANSLATE\$LOOP:  | MOV  | AL,[SI]          | ;LOAD BYTE FROM SOURCE        |
|                   | SUB  | AL,20H           | ;NORMALIZE                    |
|                   | JB   | TRANSLATE\$ERROR | ;IF LESS THAN 0, REPORT ERROR |
|                   | CMP  | AL,3FH           | ;COMPARE WITH NORMALIZED MAX. |
|                   | JA   | TRANSLATE\$ERROR | ;IF GREATER, REPORT ERROR     |
|                   | XLAT |                  | ;TRANSLATE NORMALIZED VALUE   |
|                   | MOV  | [SI],AL          | ;STORE CONVERTED DATA         |
|                   | INC  | SI               | ;ADJUST POINTERS              |
|                   | DEC  | CX               |                               |
|                   | JNZ  | TRANSLATE\$LOOP  |                               |
|                   | RET  |                  | ;GOOD RETURN WITH Z=1         |
| TRANSLATE\$ERROR: | RET  |                  | ;ERROR RETURN WITH Z=0        |

図4-27 範囲をチェックするバッファ内容の変換

変換エラーがなければ $ZF = 1$ で、1つあるいはそれ以上の変換エラーがあれば $ZF = 0$ で戻る。減算命令によって変換テーブルの大きさが $40_{16}$ バイトに制限されていることに注意。2つのCMP命令を変換テーブルの $20_{16}$ バイト前の位置を示すBXレジスタと用いれば、データを有効に利用することができる。

### 4.3 論理演算命令

8086は、次に示す通常の論理演算機能を持つ。

AND  
NOT  
OR  
XOR

さらに、オペランドのどちらも変えないAND操作を行なうTEST命令がある。

8086の論理演算命令を表4-7に示す。

図4-29は、8086の論理演算の例を示している。次の情况进行える。

1. I/Oポートは、一連のデータ・ブロックを受け取る。このデータ・ブロックは、Signetics 16進コードである。
2. I/Oポートは、キャラクタが有効になるたびに割り込みを発生する。

図4-29のルーチンは、次のようなステップで上記の情况进行処理するインタラプト・サービス・ルーチンである。

1. 情報をそのままセーブする。
2. その情報をオブジェクト・コードに変換する。
3. チェックサムを調べる。
4. 処理が終了すれば、メッセージの“メッセージ完了”ビットをセットする。

図4-29のルーチンは、図4-28のフローチャートを実行して、上記機能を行なっている。

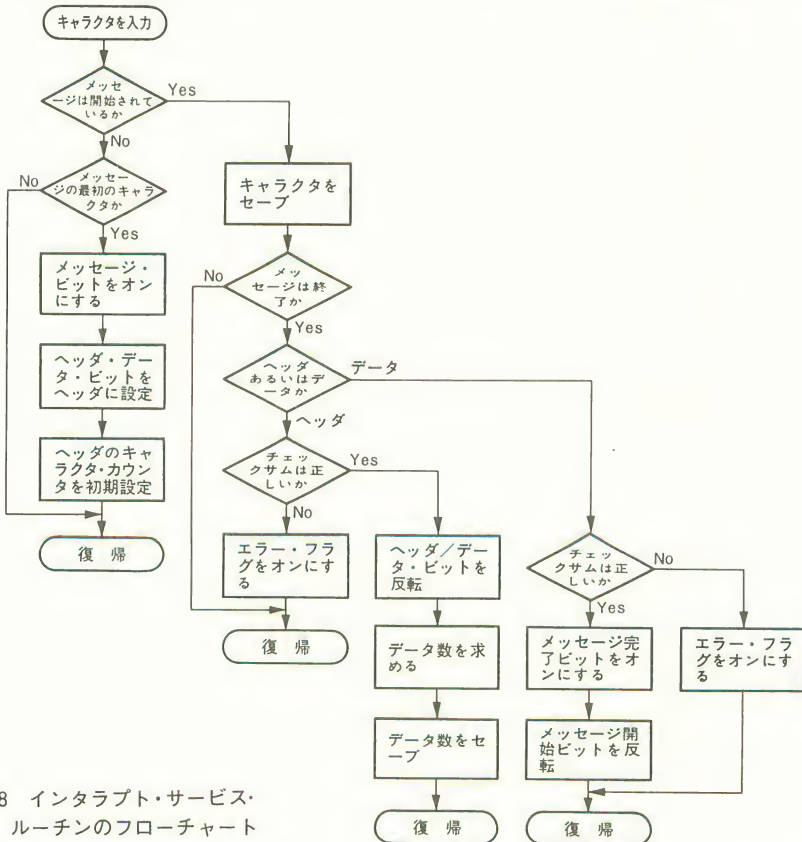


図4-28 インタラプト・サービス・ルーチンのフローチャート



表 4-7 8086の論理演算命令

| ニーモニック | オペランド                 | オブジェクト・コード                                                       | バイト             | クロック                                                    | ステータス |   |   |   |   |   |   |   | 動作内容 |                                                                                                                                                                                        |
|--------|-----------------------|------------------------------------------------------------------|-----------------|---------------------------------------------------------|-------|---|---|---|---|---|---|---|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                       |                                                                  |                 |                                                         | O     | D | I | T | S | Z | A | P |      | C                                                                                                                                                                                      |
| AND    | mem/reg1,<br>mem/reg2 | 001000dw<br>mod rrr r/m<br>(DISP)<br>(DISP)                      | 2, 3ま<br>たは4    | reg-reg: 3<br>mem-reg:<br>9 + EA<br>reg-mem:<br>16 + EA | X     |   |   |   | X | X | ? | X | X    | (mem/reg1) ← (mem/reg1) AND (mem/reg2)<br>mem/reg1で示されるメモリ位置あるいはレジスタの8<br>あるいは16ビットの内容と、mem/reg2で示されるメモ<br>リ位置あるいはレジスタの8あるいは16ビットの内容と<br>のANDをとり、mem/reg1で示されるメモリ位置ある<br>いはレジスタに結果を残す。 |
| AND    | mem/reg,<br>data      | 1000000w<br>mod 100 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (w=1のとき) | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>17 + EA                               | X     |   |   |   | X | X | ? | X | X    | (mem/reg) ← (mem/reg) AND data<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容と、8あるいは16ビットのイミデ<br>ィエイト・データとのANDをとり、mem/regで示され<br>るメモリ位置あるいはレジスタに結果をストアする。                              |
| AND    | ac, data              | 0010010w<br>kk<br>jj (w=1のとき)                                    | 2ま<br>たは3       | 4                                                       | X     |   |   |   | X | X | ? | X | X    | (ac) ← (ac) AND data<br>AL (8ビット操作) あるいはAX (16ビット操作) のレジ<br>スタと、8あるいは16ビットのイミディエイト・デー<br>タとのANDをとり、結果をALあるいはAXのレジスタに残<br>す。                                                            |
| NOT    | mem/reg               | 1111011w<br>mod 010 r/m<br>(DISP)<br>(DISP)                      | 2, 3ま<br>たは4    | reg: 3<br>mem:<br>16 + EA                               |       |   |   |   |   |   |   |   |      | (mem/reg) ← (mem/reg)<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容の1の補数をとる。                                                                                                          |
| OR     | mem/reg1,<br>mem/reg2 | 000010dw<br>mod rrr r/m<br>(DISP)<br>(DISP)                      | 2, 3ま<br>たは4    | reg-reg: 3<br>mem-reg:<br>9 + EA<br>reg-mem:<br>16 + EA | X     |   |   |   | X | X | ? | X | X    | (mem/reg1) ← (mem/reg1) OR (mem/reg2)<br>mem/reg1で示されるメモリ位置あるいはレジスタの8<br>あるいは16ビットの内容と、mem/reg2で示されるメモ<br>リ位置あるいはレジスタの8あるいは16ビットの内容と<br>のオアをとり、mem/reg1で示されるメモリ位置ある<br>いはレジスタに結果を残す。   |
| OR     | mem/reg,<br>data      | 1000000w<br>mod 001 r/m<br>(DISP)<br>(DISP)<br>kk<br>jj (w=1のとき) | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>17 + EA                               | X     |   |   |   | X | X | ? | X | X    | (mem/reg) ← (mem/reg) OR data<br>mem/regで示されるメモリ位置あるいはレジスタの8あ<br>るいは16ビットの内容と、8あるいは16ビットのイミデ<br>ィエイト・データとのオアをとり、mem/regで示され<br>るメモリ位置あるいはレジスタに結果を残す。                                   |



表 4-7 8086の論理演算命令 (続き)

| ニーモニック | オペランド                | オブジェクトコード                                         | バイト             | クロック                                                    | ステータス |   |   |   |   |   |   |   | 動作内容 |
|--------|----------------------|---------------------------------------------------|-----------------|---------------------------------------------------------|-------|---|---|---|---|---|---|---|------|
|        |                      |                                                   |                 |                                                         | O     | D | I | T | S | Z | A | P | C    |
| OR     | ac,data              | 0000110w<br>kk<br>jj (w=1のとき)                     | 2または3           | 4                                                       | X     |   |   |   | X | X | ? | X | X    |
| TEST   | mem/reg,<br>mem/reg2 | 1000010w<br>mod rrr r/m<br>(DISP)<br>(DISP)       | 2, 3または4        | reg-reg: 3<br>mem-reg:<br>9 + EA                        | X     |   |   |   | X | X | ? | X | X    |
| TEST   | mem/reg,<br>data     | 1111011w<br>mod 000 r/m<br>(DISP)<br>(DISP)<br>kk | 3, 4, 5<br>または6 | reg: 5<br>mem:<br>11 + EA                               | X     |   |   |   | X | X | ? | X | X    |
| TEST   | ac,data              | 1010100w<br>kk<br>jj (w=1のとき)                     | 2または3           | 4                                                       | X     |   |   |   | X | X | ? | X | X    |
| XOR    | mem/reg,<br>mem/reg2 | 001100dw<br>mod rrr r/m<br>(DISP)<br>(DISP)       | 2, 3または4        | reg-reg: 3<br>mem-reg:<br>9 + EA<br>reg-mem:<br>16 + EA | X     |   |   |   | X | X | ? | X | X    |
| XOR    | mem/reg,<br>data     | 1000000w<br>mod 110 r/m<br>(DISP)<br>(DISP)<br>kk | 3, 4, 5<br>または6 | reg: 4<br>mem:<br>17 + EA                               | X     |   |   |   | X | X | ? | X | X    |
| XOR    | ac,data              | 0011010w<br>kk<br>jj (w=1のとき)                     | 2または3           | 4                                                       | X     |   |   |   | X | X | ? | X | X    |

以下に示すように、Signetics オブジェクト・コードは次の要素を持つ。

1. スペースを含む、任意数のプリントされないキャラクタによるギャップ。
2. ブロック・キャラクタの開始、すなわち 1 個のコロン。
3. アドレス・フィールド、すなわち 4 個の16進キャラクタ。
4. カウント・フィールド、すなわち 0 から  $1E_{16}$  の範囲の 2 個の16進キャラクタ。
5. アドレスとカウントのフィールドのブロック・コントロール・キャラクタ、すなわち 2 つの16進キャラクタ。
6. データ・フィールド、このブロックによってロードされるメモリの数を表わすカウント・フィールドの値の 2 倍の長さを持つ。
7. ブロック・コントロール・キャラクタ。これは、2 つの16進キャラクタである。

#### Signetics オブジェクト・コード・フォーマットの例

: 0500 0A 3C 0455B024FFF01F050400 30

②   ③   ④ ⑤                      ⑥                      ⑦

- ② — ブロック・キャラクタの開始(コロン)
- ③ — ブロックの開始アドレス(H'0500')
- ④ — ブロックのバイト数(H'0A'=10)
- ⑤ — フィールド3と4のBCCバイト(H'3C')
- ⑥ — データ、1バイトにつき2バイト
- ⑦ — フィールド6のBCCバイト(H'30')

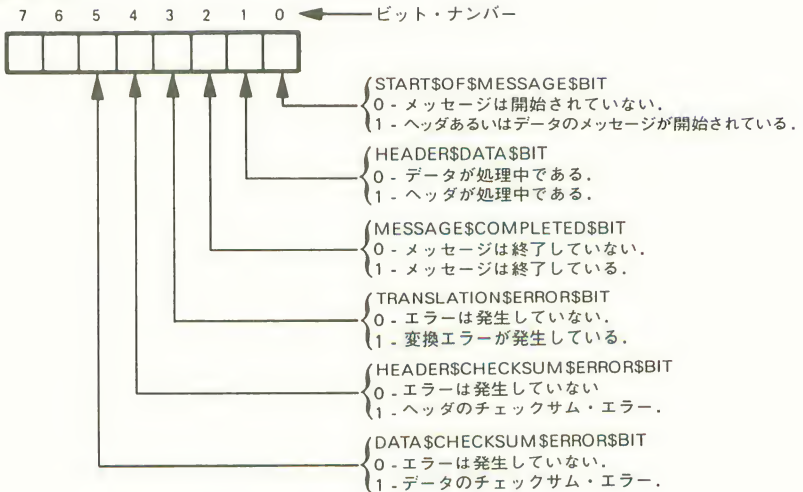
ブロック・コントロール・キャラクタ、あるいはチェックサムとして知られているものは、データ・ストリング中のデータ・バイトを操作して作られる。Signetics オブジェクト・コードのフォーマットは、2 つのブロック・コントロール・キャラクタを含む。第1のブロック・コントロール・キャラクタ⑤は、ブロックの先頭アドレスとブロック中のバイト数を含む、前の 3 つのデータ・バイトに適用される。第2のブロック・コントロール・キャラクタ⑦は、フィールド⑥のデータ・バイトのストリングに適用される。

ブロック・コントロール・キャラクタを生成するために、まずキャラクタがグリヤされ、次に関連のあるストリング中の各データ・バイトとのエクスクルーシブORが繰り返し行なわれる。エクスクルーシブORをとるたびに、結果を1ビット左へローテートする。ブロック・コントロール・キャラクタの理論を示すために、Signetics オブジェクト・コードのフォーマットの第1のブロック・コントロール・キャラクタを考える。一連のバイト 05 00 0A から次のように生成が行なわれる。

| BCC      | Data     | BCC XOR Data | 左へローテート   |
|----------|----------|--------------|-----------|
| 00000000 | 00000101 | 00000101     | 00001010  |
| 00001010 | 00000000 | 00001010     | 00010100  |
| 00010100 | 00001010 | 00011110     | 00111100  |
|          |          |              | BCC = 3CH |

図4-29のルーチンでは、データがストアされるバッファ以外に、メモリに位置する次の4つの変数を用いている。

#### STATUS\$BYTE:



**CHARACTER\$COUNT:** ヘッダあるいはデータのブロックに読まれずに残っているキャラクタ数を表す1バイト。  
 メッセージの開始によってこの変数は、NUMBER\$OF\$HEADER\$CHARACTER, すなわち8に初期設定される。ヘッダが処理されると、この変数は  $2 * (\text{データ・ブロックのオブジェクト・コードのバイト数}) + 2$  に初期設定される。

**OBJECT\$BYTES\$COUNT:** データ・ブロック中のオブジェクト・コードのバイト数を表す1バイト。  
 この変数は、ヘッダの処理が終わった後に初期設定される。

**BUFFER\$POINTER:** I/Oポートからのデータの次のバイトがストアされるべき位置を示す16ビットのオフセット・アドレス。この変数は、メッセージの開始によって初期設定され、イミディエイト・データがロードされる。これは、バッファが常に固定された位置にあることを仮定している。もしバッファの位置を変える必要があるならば、この変数はシステムあるいはユーザ・プログラムによって初期設定できる。

This page contains the equates for this program. As mentioned earlier, equates allow descriptive names to be used in a program.

```

STATUS BYTE EQUATES
STARTOF$MESSAGE$BIT      EQU  01H
HEADER$DATA$BIT          EQU  02H
MESSAGE$COMPLETED$BIT   EQU  04H
TRANSLATION$ERROR$BIT    EQU  08H
HEADER$CHECKSUM$ERROR$BIT EQU  10H
DATA$CHECKSUM$ERROR$BIT  EQU  20H

BUFFER ADDRESS EQUATES
BUFFER$ADDRESS      EQU  1000H, OFFSET ADDRESS FOR BUFFER
STARTOF$HEADER$POINTER EQU  BUFFER$ADDRESS
STARTOF$DATA$POINTER  EQU  BUFFER$ADDRESS + 8

I O EQUATE
CHARACTER$PORT      EQU  10H, I O PORT ADDRESS FOR DATA

MISCELLANEOUS EQUATES
STARTOF$MESSAGE$CHARACTER EQU  3AH
NUMBEROF$HEADER$CHARACTERS EQU  08H

DATA DEFINITION
STATUS$BYTE      DB  1
CHARACTER$COUNT DB  1
BUFFER$POINTER   DW  1
OBJECT$BYTE$COUNT DB  1
INTERRUPT$HANDLER IN  AL, CHARACTER$PORT
TEST             STATUS$BYTE, STARTOF$MESSAGE$BIT
JNZ              HEADER$OR$DATA
READ CHARACTER
HAS A MESSAGE
BEEN STARTED?

STARTOF$MESSAGE$CODE CMP  AL, STARTOF$MESSAGE$CHARACTER
JNZ              PERFORM$A$RET
START OF MESSAGE
CHARACTER?
OR                STATUS$BYTE, STARTOF$MESSAGE$BIT
HEADER$DATA$BIT   INITIALIZE
MOV              CHARACTER$COUNT, NUMBEROF$HEADER$CHARACTERS
MOV              BUFFER$POINTER, BUFFER$ADDRESS
MOVE IMMEDIATE

PERFORM$A$RET      RET
DATA TO BUFFER
POINTER

HEADER$OR$DATA     MOV  DI, BUFFER$POINTER
MOV               DI, AL
SAVE CHARACTER
INC              DI
UPDATE POINTER
MOV              DI, BUFFER$POINTER
DEC              CHARACTER$COUNT
DECREMENT AND TEST
FOR MESSAGE DONE
JNZ              PERFORM$A$RET
TEST            STATUS$BYTE, HEADER$DATA$BIT
JZ              DATA$PROCESSING
DATA$PROCESSING
MOV              CX, 0004
SET UP FOR ASCII TO HEX
CONVERSION
MOV              SI, STARTOF$HEADER$POINTER
MOV              DI, SI

HEADER$TRANSLATE$LOOP CALL  DI, CONVERT$TWO$ASCII$TO$HEX
JZ              TRANSLATION$ERROR
CONVERT TWO ASCII CHARACTERS
TO ONE HEX BYTE
MOV              DI, AL
INC              DI
DECREMENT AND TEST FOR DONE
JNZ              HEADER$TRANSLATE$LOOP
DEC              CX
SET UP FOR HEADER CHECKING
MOV              SI, STARTOF$HEADER$POINTER
XOR              AX, AX
MOV              CX, 0003

HEADER$CHECKSUM$LOOP XOR  AL, SI
ROL              AL, 1
CALCULATE BLOCK CHECKSUM
FROM CHARACTERS
INC              SI
DEC              CX
FOR CHECKSUM DONE
JNZ              HEADER$CHECKSUM$LOOP
CMP              AL, SI
COMPARE CALCULATED CHECKSUM
WITH RECEIVED CHECKSUM
JNZ              HEADER$CHECKSUM$ERROR
XOR              STATUS$BYTE, HEADER$DATA$BIT
HEADER GOOD, SWITCH TO
DATA PROCESSING
MOV              AX, [SI-2]
LOAD = OF OBJECT
CODE BYTES FROM HEADER
MOV              AX, 1
GET NUMBER OF ASCII CHARACTERS
ADD              AX, 02
ADD 2
RET              CHARACTER$COUNT, AX
SAVE FOR DATA PROCESSING

HEADER$CHECKSUM$ERROR MOV  STATUS$BYTE, HEADER$CHECKSUM$ERROR$BIT
TURN ON ERROR BIT
RET

```

図4-29 インタラプト・サービス・ルーチン



|                       |      |                                          |                                |
|-----------------------|------|------------------------------------------|--------------------------------|
| TRANSLATION\$ERROR    | MOV  | STATUS\$BYTE, TRANSLATION\$ERROR\$BIT    | .TURN ON ERROR BIT             |
| DATA\$PROCESSING      | RET  |                                          |                                |
|                       | MOV  | CX, OBJECT\$BYTES\$COUNT                 | .SET UP FOR CONVERSION         |
|                       | MOV  | SI, START\$OF\$DATA\$POINTER             | .FROM ASCII TO HEX             |
|                       | MOV  | DI, START\$OF\$DATA\$POINTER-4           |                                |
| DATA\$TRANSLATE\$LOOP | CALL | CONVERT\$TWO\$ASCII\$TO\$HEX             |                                |
|                       | JZ   | TRANSLATION\$ERROR                       |                                |
|                       | MOV  | (DI), AL                                 |                                |
|                       | INC  | DI                                       |                                |
|                       | DEC  | CX                                       | .DECREMENT AND TEST FOR        |
|                       | JNZ  | DATA\$TRANSLATE\$LOOP                    | .DONE                          |
|                       | MOV  | SI, START\$OF\$DATA\$POINTER-4           | .SET UP FOR CHECKSUM           |
|                       | XOR  | AX, AX                                   | .CALCULATION                   |
|                       | MOV  | CX, OBJECT\$BYTES\$COUNT                 |                                |
| DATA\$CHECKSUM\$LOOP  | XOR  | AL, (SI)                                 |                                |
|                       | ROL  | AL, 1                                    | .CALCULATE CHECKSUM            |
|                       | INC  | SI                                       |                                |
|                       | DEC  | CX                                       |                                |
|                       | JNZ  | DATA\$CHECKSUM\$LOOP                     |                                |
|                       | CMP  | AL, (SI)                                 | .COMPARE CALCULATED CHECKSUM   |
|                       | JNZ  | DATA\$CHECKSUM\$ERROR                    | .WITH RECEIVED CHECKSUM        |
|                       | XOR  | STATUS\$BYTE, START\$OF\$MESSAGE\$BIT    | .TURN ON                       |
|                       |      | OR MESSAGE\$COMPLETED\$BIT               | .MESSAGE COMPLETED BIT         |
|                       | RET  |                                          | .TURN OFF START OF MESSAGE BIT |
| DATA\$CHECKSUM\$ERROR | MOV  | STATUS\$BYTE, DATA\$CHECKSUM\$ERROR\$BIT | .TURN ON ERROR BYTE            |
|                       | RET  |                                          |                                |

図4-29 インタラプト・サービス・ルーチン(続き)

図4-28に示すプログラムの理論では、いくつかの仮定がある。この仮定には次のものが含まれる。

1. CPUの状態は退避されていて、復帰の際に正しく復元される。
2. セグメント・レジスタは正しい値に設定されている。
3. I/Oポートによって設定されるハードウェアのエラー・フラグは、他で処理される。
4. CONVERT\$TWO\$ASCII\$TO\$HEXという名前のサブルーチンは、DIレジスタで示される2つのASCIIキャラクタを16進バイトに変換して、その値をALに設定して戻る。

ある種類のシステム・インタラプト処理プログラムが、最初の3つの条件を備えていることを期待するのは当然といえる。もしそうでなければ、上記条件は次のようにして処理できる。

1. CPU状態の退避・復元には、図4-14と図4-15に示した命令を用いる。ただし、このルーチンはBX、DX、あるいはBPのレジスタを用いていないことに注意。したがって、これらのレジスタは退避・復元の必要がない。RET命令はIRET命令に変換しなければならない。
2. 適当なセグメント・レジスタ初期設定を行なう。
3. 入力装置のステータス・ポートを読み出して、エラー表示のためにステータス・バイトのフラグを設定する。ビット6と7はこの目的のために用いられる。

CONVERT\$TWO\$ASCII\$TO\$HEXのルーチンは、この章の後で述べる。

## 4.4 スtring・プリミティブ命令

8086のString・プリミティブ命令を表4-8に示す。各々のString・プリミティブ命令は、通常、命令ループで処理される一連の操作を行なう。String・プリミティブ命令は、指定された1つの操作を行ない、次にこの操作に含まれているポインタ・レジスタの増減を行なう。各繰り返しごとに、影響を受けるポインタ・レジスタは、1あるいは2の増減が行なわれる。ポインタ・レジスタは、ディレクション・フラグが0ならば増加し、1ならば減少する。String・プリミティブ・オペレーション・コードの最下位ビットが0ならば、ポインタ・レジスタは1の増減が行なわれる。String・プリミティブ・オペレーション・コードの最下位ビットが1ならば、影響を受けるポインタ・レジスタは2の増減が行なわれる。

次の5つのString・プリミティブ命令が存在する。

MOVSB—メモリからメモリへの8あるいは16ビットのデータを移動する。

LODSB—メモリから8あるいは16ビットのデータをALあるいはAXのレジスタにロードする。

STOSB—AL（8ビット操作）あるいはAX（16ビット操作）のレジスタの内容をメモリにストアする。

SCASB—メモリとAL（8ビット操作）あるいはAX（16ビット操作）のレジスタの比較を行なう。

CMPSB—メモリとメモリの比較を行なう。

String・プリミティブ命令は、次に示すような固定されたアドレッシング・モードを用いる。

MOVSB—データ・セグメントのSIレジスタで示されるメモリ位置から、エキストラ・セグメントのDIレジスタで示されるメモリ位置にデータを移動する。

LODSB—データ・セグメントのSIレジスタで示されるメモリ位置から、ALあるいはAXのレジスタにデータをロードする。

STOSB—ALあるいはAXのレジスタの内容を、エキストラ・セグメントのDIレジスタで示されるメモリ位置にストアする。

SCASB—エキストラ・セグメントのDIレジスタで示されるメモリ位置のデータと、ALあるいはAXのレジスタの内容を比較する。

CMPSB—データ・セグメントのSIレジスタで示されるメモリ位置のデータと、エキストラ・セグメントのDIレジスタで示されるメモリ位置のデータを比較する。

セグメント変更プレフィックスによって、SIでデータ・セグメント以外のセグメントにアクセスすることができる。セグメント変更プレフィックスは、DIレジスタには用いられない。DIレジスタは、エキストラ・セグメントをアクセスしなければならない。



表 4-8 ストリング・プリミティブ命令

| ニーモニック | オペランド | オブジェクト・コード | バイト | クロック          | ステータス |   |   |   |   |   |   |   |   |   | 動作内容                                                                                                                                                                                                                                                             |
|--------|-------|------------|-----|---------------|-------|---|---|---|---|---|---|---|---|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |       |            |     |               | O     | D | I | T | S | Z | A | P | C |   |                                                                                                                                                                                                                                                                  |
| LODS   |       | 1010110w   | 1   | 12<br>9 + 13* |       |   |   |   |   |   |   |   |   |   | $[ac] \leftarrow [(SI)], [SI] \leftarrow [SI] \pm \Delta$<br>SIで示されるメモリ位置から、AL (8ビット操作) あるいはAX (16ビット操作) のレジスタにデータを移動する。ディレクション・フラグの値に依存して、SIの増減を行なう。DELTAは、w = 0のときは1で、w = 1のときは2である。                                                                           |
| MOVS   |       | 1010010w   | 1   | 18<br>9 + 17* |       |   |   |   |   |   |   |   |   |   | $[(DI)] \leftarrow [(SI)], [SI] \leftarrow [SI] \pm \Delta$<br>$[DI] \leftarrow [DI] \pm \Delta$<br>SIで示されるメモリ位置からDIで示されるメモリ位置へ、8あるいは16ビットのデータを移動する。ディレクション・フラグの値に依存して、SIとDIの増減を行なう。DELTAは、w = 0のときは1で、w = 1のときは2である。                                            |
| STOS   |       | 1010101w   | 1   | 11<br>9 + 10* |       |   |   |   |   |   |   |   |   |   | $[(DI)] \leftarrow [ac], [DI] \leftarrow [DI] \pm \Delta$<br>AL (8ビット操作) あるいはAX (16ビット操作) のレジスタの内容を、DIレジスタで示されるメモリ位置に移動する。ディレクション・フラグの値に依存して、DIの増減を行なう。DELTAは、w = 0のときは1で、w = 1のときは2である。                                                                         |
| CMPS   |       | 1010011w   | 1   | 22<br>9 + 22* | X     |   |   | X | X | X | X | X | X | X | $[(SI)] \leftarrow [(DI)], [SI] \leftarrow [SI] \pm \Delta$<br>$[DI] \leftarrow [DI] \pm \Delta$<br>SIレジスタで示されるアドレスの8あるいは16ビットの内容から、DIレジスタで示されるアドレスの8あるいは16ビットの内容を減じる。結果はフラグを設定するために用いてから捨てられる。ディレクション・フラグの値に依存して、SIとDIの増減を行なう。DELTAは、w = 0のときは1で、w = 1のときは2である。 |
| SCAS   |       | 1010111w   | 1   | 15<br>9 + 15* | X     |   |   | X | X | X | X | X | X | X | $[ac] \leftarrow [(DI)]$<br>AL (8ビット操作) あるいはAX (16ビット操作) のレジスタから、DIレジスタで示されるアドレスの8あるいは16ビットの内容を減じる。結果はフラグを設定するために用いてから捨てられる。ディレクション・フラグの値に依存して、DIの増減を行なう。DELTAは、w = 0のときは1で、w = 1のときは2である。                                                                       |

\*REPプレフィックスが先行した場合、最初の転送は\*の数に9を加えたクロックに、以後の転送は\*の数のクロックになる。

#### 4.4.1 REPプレフィックス

REPは、ストリング・プリミティブ命令を再実行ループに変換する1バイトのプレフィックスである。

ストリング・プリミティブ命令は、それぞれループの1回の繰り返しとして実行される。ソースとディスティネーションのポインタ・レジスタのSIとDIは、ストリング・プリミティブ命令で、ソースやディスティネーションのメモリ・アドレスを与えると仮定されている。このアドレスは、ストリング・プリミティブ命令の実行に続いて、自動的に増加あるいは減少する。これにより、アドレスは、アクセスされるストリングの次の位置を示す。REPプレフィックスは、終結条件が成立するまでストリング・プリミティブ命令の実行を続けさせる条件を指定する。

MOVS, LODS, STOSのストリング・プリミティブについては、1つの終結条件が存在する。CXレジスタはカウンタとして取り扱われ、ストリング・プリミティブ命令が実行されるたびに、CXレジスタの内容はREPプレフィックスによって自動的に減少する。CXレジスタの内容が0に減少すると、ストリング・プリミティブに続く命令が実行される。

CMPSとSCASもまた、REPプレフィックスが存在するとき、CXレジスタをカウンタとして用い、MOVS, LODS, STOSについて述べたように、CXレジスタが減少して0になることが終結条件となる。さらに、CMPSとSCASは、実行を繰り返すたびにステータス・フラグを設定する。ゼロ・フラグの値は、別の終結条件として用いられる。これを可能とするために、CMPSとSCASのストリング・プリミティブは、REPプレフィックスの次の2つの形式を用いる。

1. REPZあるいはREPE. これは、ストリング・プリミティブの繰り返される実行で、ゼロ・フラグがリセットされれば終了する。
2. REPNZあるいはREPNE. これは、繰り返される実行で、ゼロ・フラグが、セットされれば終了する。

要するに、REPプレフィックスは、ストリング・プリミティブ命令の実行に、次のステップを加える。

1. CXレジスタの内容を調べる。CXが0ならば、ストリング・プリミティブに続く命令に進む。
2. 保留となっている割り込みを処理する。
3. ストリング・プリミティブ命令を一度実行する。ポインタ・レジスタのアドレスは、ストリング・プリミティブ命令実行の通常の部分として、このステップの間に増加あるいは減少する。
4. CXレジスタの内容を減じる。

5a. MOVS, LODS, あるいはSTOSについてはステップ1に進む。

5b. CMPSあるいはSCASについては、ゼロ・フラグとREPプレフィックスで指定される条件を比較する。指定されたゼロ・フラグの状態でなければ、ステップ1に

戻り、そうでなければストリング・プリミティブに続く命令を実行する。  
ストリング・プリミティブ命令は非常に有力である。

```
MOV AL,[SI]
INC SI
```

あるいは、

```
MOV AX,[SI]
INC SI
INC SI
```

などの命令は、

```
MOVSB
```

あるいは、

```
MOVSW
```

で置き換えられる。

図4-2のルーチンを考える。もしディレクション・フラグが0に設定されていれば、その一連の命令は、

```
REP MOVSW
RET
```

で置き換えられる。さらに、REPとMOVSWの命令が1バイト命令であることに注意すれば、このルーチンをコールすることの方が、プログラム中に直接

```
REP MOVSW
```

を挿入するよりも損失が大きくなる。

図4-9のバッファ初期設定ルーチンは、

```
REP STOSB
```

で置き換えられる。この置き換えでは、ディレクション・フラグが0に設定されていることを仮定している。

練習に、読者は、この章の初めに示した他のプログラムを調べて、ストリング・プリミティブの使用が可能な例を見つけられたい。

バイトより成る2つのストリングを比較する、図4-1に示されているプログラムの変形を考える。完全に書けば、プログラムは次のようになる。

|              |     |             |                           |
|--------------|-----|-------------|---------------------------|
| COMP\$BYTES: | MOV | AL,[SI]     | :LOAD BYTE FROM SOURCE    |
|              | CMP | [DI],AL     | :COMPARE WITH DESTINATION |
|              | JZ  | EQUAL       | :TEST FOR SIMILAR BYTES   |
|              | INC | SI          | :ADJUST POINTERS          |
|              | INC | DI          |                           |
|              | DEC | CX          | :DECREMENT NUMBER TO MOVE |
|              | JNZ | COMP\$BYTES | :LOOP IF NOT DONE         |

図4-30 8ビットのバッファとバッファの比較

CMPは、レジスタとレジスタ、レジスタとメモリ、あるいはメモリとレジスタで、バイトあるいはワードの比較を行なう。図4-30では、同一のバイトを探して、2つのメモリ・バッファの比較が行なわれる。ポインタ・レジスタのSIとDIは、それぞれソースとデスティネーションのアドレスを示す。レジスタのSI、DI、さらにCXは、CMP SBとCMP SWのストリング・プリミティブで用いられるレジスタなので、図4-30では慎重に選択されている。この結果、図4-30に示すプログラムの全体を次のもので置き換えることが可能となる。

|      |       |
|------|-------|
| REPZ | CMPSB |
| JZ   | EQUAL |

図4-31 別の8ビットのバッファとバッファの比較

## 4.5 プログラム・カウンタ制御命令

8086の命令のこのグループは、無条件にプログラム・カウンタの内容を変え、さらにある場合には、コード・セグメントの内容も同様に変更する。命令の要約を表4-9に示す。

CALL命令は、プログラムからサブプログラムに制御を移すために用いられる。サブプログラムは、カレント・コード・セグメントあるいは命令で指定されるコード・セグメントに存在する。サブプログラムのアドレスは、命令でイミディエイト・アドレスとして与えられるか、あるいはメモリまたはレジスタにストアされている。8086には、次の4つの可能なCALL命令がある。

|                   | カレント・コード・セグメント | 命令指定の<br>コード・セグメント |
|-------------------|----------------|--------------------|
| イミディエイト・アドレス      | CALL disp16    | CALL addr          |
| メモリあるいはレジスタ内のアドレス | CALL mem/reg   | CALL mem           |

CALL disp16 は、符号付き16ビット数値がプログラム・カウンタに加算されるただ1つの命令である。他の3つのCALL命令は、メモリあるいはレジスタからプログラム・カウンタに直接データを移動する。以前のコード・セグメントやプログラム・カウンタの内容はスタックにプッシュされて、退避される。

RETURN命令は、サブプログラムからそれを呼び出したプログラムに制御を戻すために用いられる。サブプログラムを終結させるRETURN命令は、サブプログラムを呼び出すCALLの逆の操作を行なう。RETURN命令が実行されると、プログラム・カウンタと必要ならばコード・セグメント・レジスタに、スタックからデータがポップされる。さらに、RETURN命令は、スタック・ポインタにディスプレイメントを加えることもでき



表4-9 プログラム・カウンタ制御命令

| ニーモニック | オペランド                | オブジェクトコード                             | バイト       | クロック                         | ステータス |   |   |   |   |   |   |   | 動作内容 |                                                                                                                                                                                                                                                           |
|--------|----------------------|---------------------------------------|-----------|------------------------------|-------|---|---|---|---|---|---|---|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                      |                                       |           |                              | O     | D | I | T | S | Z | A | P |      | C                                                                                                                                                                                                                                                         |
| CALL   | addr                 | 9A<br>kk<br>jj<br>hh<br>gg            | 5         | 28                           |       |   |   |   |   |   |   |   |      | $(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (PC),$<br>$(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (CS),$<br>$(PC) \leftarrow addr$ (オフセット部), $(CS) \leftarrow addr$ (セグメント部)<br>他のコード・セグメント内のサブルーチンをコールする。<br>新しいオフセット・アドレスはjkkに、新しいセグメント・アドレスはgghhになる。 |
| CALL   | disp16               | E8<br>kk<br>jj                        | 3         | 19                           |       |   |   |   |   |   |   |   |      | $(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (PC)$<br>$(PC) \leftarrow (PC) + disp16$<br>現在のコード・セグメント内のサブルーチンをコールする。                                                                                                                                      |
| CALL   | mem<br>(SEG<br>+ PC) | FF<br>mod 011 r/m<br>(DISP)<br>(DISP) | 2. 3バイトは4 | 37 + EA                      |       |   |   |   |   |   |   |   |      | $(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (PC),$<br>$(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (CS),$<br>$(PC) \leftarrow [mem], (CS) \leftarrow [mem + 2]$<br>他のコード・セグメント内のサブルーチンをコールする。memで示されるメモリ位置の16ビットの内容がPCに移動し、続くメモリ・ワードの内容がCSレジスタにロードされる。   |
| CALL   | mem/reg<br>(PCのみ)    | FF<br>mod 010 r/m<br>(DISP)<br>(DISP) | 2. 3バイトは4 | 21 + EA<br>(mem)<br>16 (reg) |       |   |   |   |   |   |   |   |      | $(SP) \leftarrow (SP) - 2, ((SP)) \leftarrow (PC)$<br>$(PC) \leftarrow [mem/reg]$<br>現在のコード・セグメント内のサブルーチンをコールする。mem/regで示されるメモリ位置あるいはレジスタの16ビットの内容がPCに移動する。                                                                                               |
| RET    |                      | C3                                    | 1         | 16                           |       |   |   |   |   |   |   |   |      | $(PC) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2$<br>現在のコード・セグメント内の呼び出し元プログラムにリターンする。                                                                                                                                                                    |
| RET    |                      | CB                                    | 1         | 24                           |       |   |   |   |   |   |   |   |      | $(PC) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2,$<br>$(CS) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2$<br>他のコード・セグメントの呼び出し元プログラムにリターンする。                                                                                                               |
| RET    | disp16               | C2<br>kk<br>jj                        | 3         | 20                           |       |   |   |   |   |   |   |   |      | $(PC) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2 + disp16$<br>現在のコード・セグメント内の呼び出し元プログラムにリターンし、スタック・ポインタをdisp16だけ調整する。                                                                                                                                     |
| RET    | disp16               | CA<br>kk<br>jj                        | 3         | 23                           |       |   |   |   |   |   |   |   |      | $(PC) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2,$<br>$(CS) \leftarrow ((SP)), (SP) \leftarrow (SP) + 2 + disp16$<br>他のコード・セグメントの呼び出し元プログラムにリターンし、スタック・ポインタをdisp16だけ調整する。                                                                                |

表 4-9 プログラム・カウンタ制御命令 (続き)

| ニーモニック | オペランド                | オブジェクト・コード                            | バイト          | クロック                         | ステータス |   |   |   |   |   |   |   | 動作 内 容                                                                                                        |
|--------|----------------------|---------------------------------------|--------------|------------------------------|-------|---|---|---|---|---|---|---|---------------------------------------------------------------------------------------------------------------|
|        |                      |                                       |              |                              | O     | D | I | T | S | Z | A | P |                                                                                                               |
| JMP    | addr                 | EA<br>kk<br>jj<br>hh                  | 5            | 15                           |       |   |   |   |   |   |   |   | (PC) ←addr(オフセット部), (CS) ←addr(セグメント部)<br>他のコード・セグメントにジャンプする。新しいオフセ<br>ット・アドレスはkkに、新しいセグメント・アドレスは<br>gghhになる。 |
| JMP    | disp                 | EB<br>gg                              | 2            | 15                           |       |   |   |   |   |   |   |   | (PC) ←(PC) + disp<br>プログラム・カウンタ相対のジャンプを行なう。                                                                   |
| JMP    | disp 16              | E9<br>kk<br>jj                        | 3            | 15                           |       |   |   |   |   |   |   |   | (PC) ←(PC) + disp 16<br>現在のコード・セグメント内でジャンプを行なう。                                                               |
| JMP    | mem<br>(SEG<br>+ PC) | FF<br>mod 101 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは4 | 24 + EA                      |       |   |   |   |   |   |   |   | (PC) ←[mem], (CS) ←(mem + 2)<br>他のコード・セグメント内へジャンプする。memで示さ<br>れるメモリ位置の内容をPCに移動し、続くメモリ位置の<br>内容をCSレジスタに移動する。   |
| JMP    | mem/reg<br>(PCのみ)    | FF<br>mod 100 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは4 | 16 + EA<br>(mem)<br>11 (reg) |       |   |   |   |   |   |   |   | (PC) ←[mem/reg]<br>現在のコード・セグメント内のメモリ位置にジャンプす<br>る。mem/regで示されるメモリ位置あるいはレジスタの<br>内容をPCに移動する。                   |



る。これにより、サブプログラムが操作するスタック上のパラメータの受け渡しができるように、RETURN 命令がスタック・ポインタを調整することが可能になる。8086には、次の4つのRETURN 命令がある。

|                        | PCへのポップ    | CSとPCへのポップ |
|------------------------|------------|------------|
| 通常のリターン                | RET        | RET        |
| スタックへの<br>ディスプレイメントの加算 | RET disp16 | RET disp16 |

8086は、条件付きコールや条件付きリターンの命令を持っていないことに注意。8080に対応する命令を実現するには、CALL あるいはRETURN の命令を条件付きジャンプ命令と共に用いる必要がある。たとえば、次の8080の命令は、

```

                CNZ      SUB$PROGRAM      CALL SUB$PROGRAM if ZF = 0
NEXT$INSTRUCTION:  ORA      A

```

以下の8086の命令で置き換えられる。

```

                JZ      NEXT$INSTRUCTION ;JUMP TO NEXT$INSTRUCTION if ZF = 1
                CALL   SUB$PROGRAM      ;JUMP TO SUB$PROGRAM if ZF = 0
NEXT$INSTRUCTION:  OR      AX,BX

```

8086のジャンプ命令は表4-9に示されている。8086のジャンプ命令は一般に、8086のCALL 命令と同様の変形を有する。さらにジャンプ命令には、

JMP disp

の形式があり、これは、3 バイトの JMP disp16 の命令に対して、2 バイトのオブジェクト・コードを持つ。JMP disp は相対分岐で、プログラム・カウンタに8ビット符号付き2進数のディスプレイメントを加算する。これにより、1 ないし 127 バイト以内のジャンプ命令が可能になる。この章の多くのプログラム例では、ジャンプ命令を用いてその使用法を示している。

#### 4.5.1 条件付きジャンプ命令

いろいろな条件に基づいてプログラム・カウンタの内容を変更する、8086の命令を表4-10に示す。

表4-11には、一般に用いられる算術演算の比較をあげ、8086でそれがどのように得られるかを示してある。

一般に、大きいとか小さいとかいうのは符号付き操作に用いられる形容詞であり、上（上位）あるいは下（下位）は符号なし操作に用いられる形容詞である。

CXレジスタの内容を減少して、必要に応じてプログラム・カウンタの内容を変更する8086の命令を表4-12に示す。これらの命令は、一般にLOOP 命令と呼ばれる。練習として、この章の前節を検討して、

```

                DEC      CX
                JNZ     label

```

の命令構成を、

```
LOOP    label
```

で置き換えてみよ。

1つの置き換えは、1バイトのオブジェクト・コードの節約になる。さらに、1回の実行につき、1クロックが節約される。100回の繰り返しを行なうループの一度の実行について、これは、100クロックあるいは5MHzの8086で20マイクロ秒の節約を意味している。

JCXZの命令は、このグループの命令の中では特異な存在で、フラグ・レジスタの内容に基づいて分岐するのではなくて、CXレジスタが0ならばジャンプする。JCXZ命令は、LOOP命令と共にCXレジスタと関係を持つので、LOOP命令と共に表4-12にも示してある。

#### (1) LOOP命令

LOOP命令は、DEC CXとJNZの命令を兼ねている。たとえば、図4-1の命令は、次のように書き改めることができる。

```
MOVE$BYTES:      MOV     AL, [SI]
                   MOV     [DI], AL
                   INC     SI
                   INC     DI
                   LOOP    MOVE$BYTES
```

以後、すべての命令において、LOOP命令は、

```
DEC     CX
JNZ     label
```

の命令で置き換えられる。

## 4.6 プロセッサ制御命令

フラグに作用し、8086の外部インターフェイスをいろいろな面から制御する、8086の命令を表4-13に示す。

表4-10 零件付托分配合今

[illegible]



表4-11 符号付きと符号なしの比較命令

|   | 符号付き               |                     | 符号なし      |                  |
|---|--------------------|---------------------|-----------|------------------|
| = | .EQ. JE<br>またはJZ   | 等しい、またはO            | JEまたはJZ   | 等しい、またはO         |
| ≠ | .NE. JNE<br>またはJNZ | 等しくない、またはOでない       | JNEまたはJNZ | 等しくない、またはOでない    |
| > | .GT. JG<br>またはJNLE | 大きい、または小さくもなく等しくもない | JAまたはJNBE | 上、または下でもなく等しくもない |
| ≥ | .GE. JGE<br>またはJNL | 大きいか等しい、または小さくはない   | JAEまたはJNB | 上か等しい、または下ではない   |
| < | .LT. JL<br>またはJNGE | 小さい、または大きくもなく等しくもない | JBまたはJNAE | 下、または上でもなく等しくもない |
| ≤ | .LE. JLE<br>またはJNG | 小さいか等しい、または大きくはない   | JBEまたはJNA | 下か等しい、または上ではない   |





表4-13 プロセッサ制御命令

[illegible]

## 4.7 入出力命令

8086の入力と出力の機能を行なう命令を、表4-14に示す。

8086でのメモリ・マップド・アドレッシングは、I/Oポートのアドレス指定に重要な利点を持つ。メモリ・マップドI/Oアドレスのコード化がどのように行なわれるかによって、ストリング・プリミティブ命令はメモリに対して行なわれる繰り返し操作を可能にする。I/Oポート・アドレッシングが用いられている図4-32と、メモリ・マップド・アドレッシングが用いられている図4-33を比較せよ。この両方のルーチンはブロックのデータを出力する。ブロック中のバイト数はCXレジスタに含まれている。図4-32のルーチンは、DIレジスタで示されるブロックをIO\$PORTに出力する。図4-33のルーチンは、SIレジスタで示されるブロックをDIレジスタで示されるメモリ・マップドI/Oポートに移動する。I/Oポートのアドレスは、直接に指定されるか、あるいはDXレジスタに保持される。8ビットのアドレスは直接に指定され、16ビットのI/Oポート・アドレスはDXレジスタによって指定される。

|                  |       |                 |
|------------------|-------|-----------------|
| OUTPUT\$A\$BYTE: | LODSB |                 |
|                  | OUT   | IO\$PORT,AL     |
|                  | LOOP  | OUTPUT\$A\$BYTE |
|                  | RET   |                 |

図4-32 I/Oポート・アドレス指定によるブロックI/O

|     |      |
|-----|------|
| REP | MOVS |
|     | RET  |

図4-33 メモリ・マップ・アドレス指定によるブロックI/O

MOVSはSIとDI中のアドレスを自動的に増加あるいは減少させるので、ブロックのメモリ・アドレスはメモリ・マップドI/Oポートに割り付けられる必要のあることに注意。

表4-14 8086のI/O 命令

| ニーモニック | オペランド   | オブジェクト・コード     | バイト | クロック | ステータス |   |   |   |   |   |   |   |   |  | 動作 内 容                                                                        |
|--------|---------|----------------|-----|------|-------|---|---|---|---|---|---|---|---|--|-------------------------------------------------------------------------------|
|        |         |                |     |      | O     | D | I | T | S | Z | A | P | C |  |                                                                               |
| IN     | ac,DX   | 1110110w       | 1   | 8    |       |   |   |   |   |   |   |   |   |  | {ac}←{PORTDX}<br>DXレジスタで示されるI/Oポートから,AL(8ビット操作)あるいはAX(16ビット操作)のレジスタに入力する.     |
| IN     | ac,port | 1110010w<br>kk | 2   | 10   |       |   |   |   |   |   |   |   |   |  | {ac}←{port}<br>命令の2番目のバイトで示されるI/Oポートから,AL(8ビット操作)あるいはAX(16ビット操作)のレジスタに入力する.   |
| OUT    | ac,DX   | 1110011w       | 1   | 8    |       |   |   |   |   |   |   |   |   |  | {PORTDX}←{ac}<br>DXレジスタで示されるI/Oポートに,AL(8ビット操作)あるいはAX(16ビット操作)のレジスタの内容を出力する.   |
| OUT    | ac,port | 1110111w<br>kk | 2   | 10   |       |   |   |   |   |   |   |   |   |  | {port}←{ac}<br>命令の2番目のバイトで示されるI/Oポートに,AL(8ビット操作)あるいはAX(16ビット操作)のレジスタの内容を出力する. |

## 4.8 インタラプト命令

ソフトウェア・インタラプト命令、オーバーフローに関するインタラプト命令、インタラプトから復帰する命令を表4-15に示す。

ソフトウェア・インタラプト命令は、次の主要な目的に用いられる。

1. プログラムのデバッグ. 1 バイトのソフトウェア・インタラプト命令は、そのアドレスが 0000C<sub>16</sub> にあるルーチン呼び出す。通常、このルーチンはデバッグ・パッケージの一部であり、ブレークポイントの処理に用いられる。
2. そのアドレスがメモリの最初の1024バイトに存在するサブルーチンの呼び出し. 2 バイトのソフトウェア・インタラプト命令が用いられると、そのアドレスがメモリの最初の1024バイトにある 256 個のサブルーチンの 1 つが呼び出される。

ソフトウェア・インタラプト命令は、セグメント間CALLで用いられるプログラム・メモリの5バイトと比較して、1あるいは2バイトのプログラム・メモリを用いる利点を持っている。さらに、ソフトウェア・インタラプトは、フラグ・レジスタをスタックに自動的にセーブする。これは多くの場合、望ましい特徴である。小さな欠点は、ソフトウェア・インタラプトによってルーチンが呼ばれると、このルーチンからはIRET命令によって復帰する必要がある、これはRET命令より多くの時間を要することがあげられる。



表4-15 8086のインタラプト命令

| ニーモニック | オペランド | オブジェクト・コード                 | バイト | クロック           | ステータス |   |   |   |   |   |   |   |   |   | 動作内容                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------|-------|----------------------------|-----|----------------|-------|---|---|---|---|---|---|---|---|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |       |                            |     |                | O     | D | I | T | S | Z | A | P | C |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| INT    |       | 1100110v<br>kk (v = 1 のとき) |     |                |       | 0 | 0 | 0 | 0 |   |   |   |   |   | $[SP] \leftarrow [SP] - 2, [(SP)] \leftarrow \{FLAGS\}, [IF] \leftarrow 0, [TF] \leftarrow 0,$<br>$[SP] \leftarrow [SP] - 2, [(SP)] \leftarrow \{CS\}, [SP] \leftarrow [SP] - 2,$<br>$[(SP)] \leftarrow \{PC\}, \{CS\} \leftarrow \{\text{ベクタ(セグメント部)}\},$<br>$[PC] \leftarrow \{\text{ベクタ(オフセット部)}\}$<br>v = 0 のとき, $\text{ベクタ(オフセット部)} = \{0000C_{16}\}$<br>$\text{ベクタ(セグメント部)} = \{0000E_{16}\}$<br>v = 1 のとき, $\text{ベクタ(オフセット部)} = \{(kk*4)\}$<br>$\text{ベクタ(セグメント部)} = \{(kk*4) + 2\}$<br>ソフトウェア・インタラプトを発生させる。 |
| INTO   |       | CE                         | 1   | 53<br>(OF = 1) |       | 0 | 0 | 0 | 0 |   |   |   |   |   | $\{OF\} = 1$ のとき $[SP] \leftarrow [SP] - 2, [(SP)] \leftarrow \{FLAGS\}, [IF] \leftarrow 0,$<br>$[TF] \leftarrow 0, [SP] \leftarrow [SP] - 2, [(SP)] \leftarrow \{CS\}, [SP] \leftarrow [SP] - 2,$<br>$[(SP)] \leftarrow \{PC\}, \{CS\} \leftarrow \{00012_{16}\}, [PC] \leftarrow \{00010_{16}\}.$<br>オーバーフロー・フラグが 1 ならば、オーバーフロー処理<br>のベクタによってソフトウェア・インタラプトを発生さ<br>せる。そうでなければ、次に続く命令を実行する。                                                                                                                          |
| IRET   |       | CF                         | 1   | 32<br>(OF = 0) | X     | X | X | X | X | X | X | X | X | X | $[PC] \leftarrow [(SP)], [SP] \leftarrow [SP] + 2, \{CS\} \leftarrow [(SP)], [SP] \leftarrow [SP] + 2, \{FLAGS\} \leftarrow [(SP)], [SP] \leftarrow [SP] + 2$<br>インタラプト・サービス・ルーチンからリターンする。                                                                                                                                                                                                                                                                                                                           |

## 4.9 ローテートとシフトの命令

ローテートとシフトを行なう、8086の命令を表4-16に示す。

ローテートとシフトの命令は、ビットを調べる操作に多く用いられる。これらの命令は、種々のビット・パターンを調べるために、単独で、あるいは論理演算操作と共に用いられる。レジスタの最下位ビットを調べるには、

```
ROR reg, 1
```

の命令の方が、

```
AND reg, 01H
```

あるいは、

```
TEST reg, 01H
```

の命令よりも1サイクル速い。ゼロ・フラグが意味を持つANDあるいはTESTの命令に対して、ROR命令はキャリー・フラグを調べる。16ビットのポインタやレジスタの最下位ビットを調べるためには、

```
ROR reg, 1
```

を用いる。この命令ではオブジェクト・コードが1バイト節約され、処理は、

```
AND reg, 0001H
```

あるいは、

```
TEST reg, 0001H
```

の命令よりも1サイクル速い。ただし、ROR命令はレジスタの内容を変えるのに対し、TEST命令は非破壊であることに注意。

上に述べたように、8ビット・レジスタあるいは16ビット・レジスタの最上位ビットは、ROR命令をROL命令で置き換えることによって調べられる。

ローテートとシフトは算術演算操作を行なう。算術的なシフト操作は、乗算と除算を行なうために用いることができる。図4-34のルーチンは、2桁のASCIIキャラクタを16進数に変換する。このルーチンは、SIレジスタが2つのキャラクタ（上位バイトが最初に位置する）を示し、結果をALレジスタに入れて復帰することを仮定している。またこのルーチンでは、変換されるバイトが、0から9あるいはAからFの範囲にあることを保証している。もしこの範囲外のキャラクタであれば復帰時にゼロ・フラグを1にし、そうでなければゼロ・フラグを0にする。

|                              |       |                         |                      |
|------------------------------|-------|-------------------------|----------------------|
| CONVERT\$TWO\$ASCII\$TO\$HEX | PROC  | NEAR                    |                      |
|                              | PUSH  | CX                      |                      |
|                              | LODSB |                         | ;LOAD FROM SI TO AL  |
|                              | CALL  | CONVERT\$ASCII\$TO\$HEX |                      |
|                              | JZ    | TRANSLATION\$ERROR      |                      |
|                              | MOV   | CL,4                    | ;SET UP FOR ROTATE   |
|                              | SAL   | AL,CL                   | ;ROTATE FOUR TIMES   |
|                              | MOV   | AH,AL                   | ;SAVE IN AH          |
|                              | LODSB |                         |                      |
|                              | CALL  | CONVERT\$ASCII\$TO\$HEX |                      |
|                              | JZ    | TRANSLATION\$ERROR      |                      |
|                              | OR    | AL,AH                   | ;CREATE THE HEX BYTE |
|                              | OR    | AH,OFFH                 | ;TURN ZF=0           |
|                              | POP   | CX                      |                      |
|                              | RET   |                         |                      |
| TRANSLATION\$ERROR:          | RET   |                         | ;ZF IS KNOWN TO BE 1 |
| CONVERT\$TWO\$ASCII\$TO\$HEX | ENDP  |                         |                      |
| CONVERT\$ASCII\$TO\$HEX      | PROC  | NEAR                    |                      |
|                              | SUB   | AL,30H                  |                      |
|                              | JL    | TRANNNY\$ERROR          |                      |
|                              | CMP   | AL,0AH                  | ;IS IT 0 - 9         |
|                              | JL    | DONE                    |                      |
|                              | SUB   | AL,07H                  | ;ADJUST FOR A - F.   |
|                              | CMP   | AL,10H                  | ;IS IT MORE?         |
|                              | JGE   | TRANNNY\$ERROR          |                      |
| DONE:                        | RET   |                         |                      |
| TRANNNY\$ERROR:              | XOR   | AH,AH                   |                      |
|                              | RET   |                         |                      |
| CONVERT\$ASCII\$TO\$HEX      | ENDP  |                         |                      |

図4-34 ASCII表示の2桁を16進数に変換するルーチン

表4-16 8086のシフトとローテートの命令


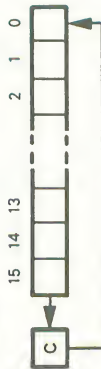
| ニーモニック | オペランド             | オブジェクトコード                                   | バイト          | クロック                                                                                                            | ステータス |                |   |   |   |   |   |   | 動作内容 |   |                                                                                                                                                                                                                                                                                                                             |
|--------|-------------------|---------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------------------|-------|----------------|---|---|---|---|---|---|------|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                   |                                             |              |                                                                                                                 | O     | D <sup>v</sup> | I | T | S | Z | A | P |      | C |                                                                                                                                                                                                                                                                                                                             |
| RCL    | mem/reg,<br>count | 110100vw<br>mod 010 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは4 | count = 1,<br>reg: 2<br>mem:<br>15 + EA<br>count<br>= [CL]<br>reg:<br>8 + 4*[CL]<br>mem:<br>20 + EA +<br>4*[CL] | X     |                |   |   |   |   |   |   |      |   | mem/regで示されるメモリ位置あるいはレジスタの内容を、キャリー・フラグと共に左へローテートする。ローテートのビット数はcountで決まり、1(v=0)あるいはCLレジスタの内容(v=1)となる。ローテートは次のように行なわれる。<br><br>w=0のとき<br><br><br>w=1のとき<br> |

表4-16 8086のシフトとローテートの命令 (続き)


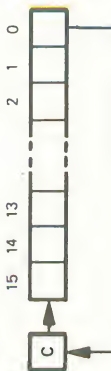
| ニーモニック | オペランド             | オブジェクト・コード                                  | バイト          | クロック                                                                                                            | ステータス |   |   |   |   |   |   |   | 動作 内 容 |   |                                                                                                                                                                                                                                                                                                                                   |
|--------|-------------------|---------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------------------|-------|---|---|---|---|---|---|---|--------|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                   |                                             |              |                                                                                                                 | O     | D | I | T | S | Z | A | P |        | C |                                                                                                                                                                                                                                                                                                                                   |
| RCR    | mem/reg,<br>count | 110100vw<br>mod 011 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは4 | count = 1,<br>reg: 2<br>mem:<br>15 + EA<br>count<br>= [CL]<br>reg:<br>8 + 4*[CL]<br>mem:<br>20 + EA<br>+ 4*[CL] | X     |   |   |   |   |   |   |   |        |   | mem/regで示されるメモリ位置あるいはレジスタの内容を、キャリー・フラグと共に右へローテートする。ローテートのビット数はcountで決まり、1(v=0) あるいはCLレジスタの内容 (v=1) となる。ローテートは次のように行われる。<br><br>w= 0 のとき<br><br><br>w= 1 のとき<br> |



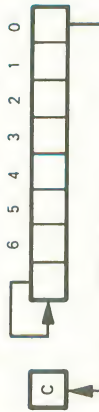
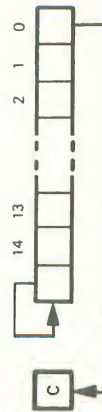
表4-16 8086のシフトとローテートの命令(続き)

| ニーモニック                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | オペランド             | オブジェクトコード                                   | バイト           | クロック                                                                                                            | ステータス |   |   |   |   |   |   |   | 動作内容 |   |                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|---------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|-------|---|---|---|---|---|---|---|------|---|----------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                   |                                             |               |                                                                                                                 | O     | D | I | T | S | Z | A | P |      | C |                                                                                                                                              |
| ROL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | mem/reg;<br>count | 110100vw<br>mod 000 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは 4 | count = 1;<br>reg: 2<br>mem:<br>15 + EA<br>count<br>= [CL]<br>reg:<br>8 + 4*[CL]<br>mem:<br>20 + EA<br>+ 4*[CL] | X     |   |   |   |   |   |   |   |      |   | mem/regで示されるメモリ位置あるいはレジスタの内容を左へローテートする。オペランドの最上位ビットはキャリー・フラグにはいる。ローテートのビット数はcountで決まり, 1 (v = 0) あるいはCLレジスタの内容 (v = 1) となる。ローテートは次のように行なわれる。 |
| <div><div>w = 0 のとき</div><div><div>C</div><div><div>7</div><div>6</div><div>5</div><div>4</div><div>3</div><div>2</div><div>1</div><div>0</div></div><div></div></div></div> <div><div>w = 1 のとき</div><div><div>C</div><div><div>15</div><div>14</div><div>13</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div></div> |                   |                                             |               |                                                                                                                 |       |   |   |   |   |   |   |   |      |   |                                                                                                                                              |





表4-16 8086のシフトとローテートの命令(続き)

| ニーモニック | オペランド             | オブジェクト・コード                                  | バイト           | クロック                                                                                                            | ステータス |   |   |   |   |   |   |   | 動作 内 容 |   |                                                                                                                                                                                                                                                                                                                                              |
|--------|-------------------|---------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|-------|---|---|---|---|---|---|---|--------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                   |                                             |               |                                                                                                                 | O     | D | I | T | S | Z | A | P |        | C |                                                                                                                                                                                                                                                                                                                                              |
| SAR    | mem/reg.<br>count | 110100vw<br>mod 111 r/m<br>(DISP)<br>(DISP) | 2, 3ま<br>たは 4 | count = 1,<br>reg: 2<br>mem:<br>15 + EA<br>count<br>= [CL]<br>reg:<br>8 + 4*[CL]<br>mem:<br>20 + EA<br>+ 4*[CL] | X     |   |   |   |   | X | X | ? | X      | X | mem/regで示されるメモリ位置あるいはレジスタの内容を、右へシフトする。最上位ビットの値を残すために、オペランドの符号を拡張する。シフトのビット数はcountで決まり、1(v=0)あるいはCLレジスタの内容(v=1)となる。シフトは次のように行われる。<br><br>w = 0 のとき<br><br><br>w = 1 のとき<br> |



# 第5章

## ソフトウェア開発

ソフトウェア開発過程での大きな補助となる、次の3つの道具がある。

### ●エディタ

エディタは、ソース・コードの入力や修正に用いられる。ソース・コードは普通、フロッピー・ディスク、ハード・ディスク、または非常の場合にペーパー・テープなどの、ある形式のマス記憶にセーブされる。ソース・コードは通常、マス記憶ではソース・ファイルと呼ばれるファイルの形式で構成されている。エディタは、ソース・ファイルのアクセス方法に依存して、いくつかの方法でソース・ファイルの作成と処理を行なう。たとえば、マグネティック・テープ中のソース・ファイルは、フロッピー・ディスク中のソース・ファイルと同じ方法では処理されない。ユーザは一般に、ビデオ・ターミナルからコマンドを入力して、エディタの機能を行なわせる。

### ●アセンブラ

アセンブラは、ソース・コードをオブジェクト・コードに翻訳するために用いられる。アセンブラは、ほとんどの場合、エディタによって作られたソース・ファイルを読み出して、ソース・コードを翻訳し、オブジェクト・ファイルと呼ばれるファイルの形式で、マス記憶にオブジェクト・コードを書き込む。アセンブラはまた、リスティング・ファイルと呼ばれる、ソース・コードとオブジェクト・コードを含むファイルや、シンボル・ファイルと呼ばれる、ソース・コードで用いられているすべてのラベルや変数名を含むファイルなど、付加的なものも作成する。リスティング・ファイルは普通、印刷されて、デバッグの過程で参照される。

### ●デバッガ

デバッガは、オブジェクト・コード中のエラー検出を援助するために用いられる。オブジェクト・ファイルがマス記憶からロードされることを要求できる位置に、デバッガはアセンブラによって生成されたオブジェクト・コードとロードされるか、あるいは単独にロードされる。代表的なデバッガでは、オブジェクト・コードの実行を制御し、メモリとレ



ジスタの内容を見ることができる。

以上述べた3つのプログラムは、開発過程に本質的で主要な道具である。その他の有用な道具には、リンカとローダがある。リンカは、複数のサブプログラムを1つのプログラムに結合するために用いられる。リンカは、サブプログラムからの外部参照を解決する。外部参照は、1つのモジュール内の命令が、他のモジュールで定義されているシンボル(ラベルまたは変数名)を参照するときに生じる。ローダは、マス記憶からメモリにオブジェクト・コードを移すために用いられる。

これを論じる目的で、ソフトウェア開発過程で用いる仮想システムを考える。これには、次のハードウェア要素が含まれている。

- CPU
- RAM
- フロッピー・ディスク装置
- CRTターミナル
- プリンタ

これらの要素の接続を図5-1に示す。

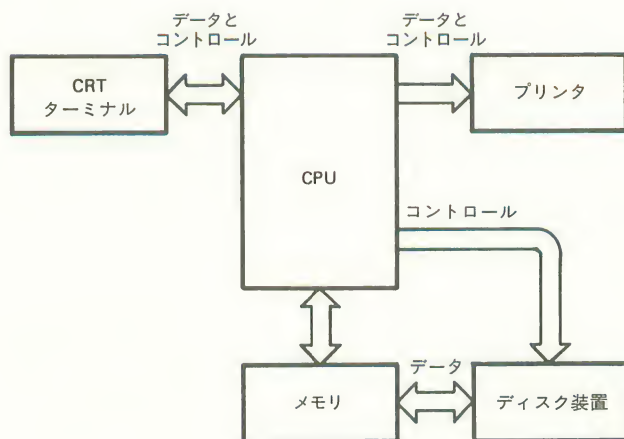


図5-1 仮想の開発システム

エディタ、アセンブラ、そしてデバッガの解説を通して、これら支援プログラムは上記のシステムで実行されることを仮定している。さらに、各支援プログラムの解説に、支援プログラムが備えている代表的な機能を示す例で、支援プログラムの仮想的コマンド言語を用いている。このシステムとコマンド言語は単なる例であって、実在しないことを強調しておく。

## 5.1 エディタ

多くのエディタは、次の作業を組み合わせることによってその機能を果たしている。

- マス記憶からメモリにデータを読み込む。
- ユーザのコマンドに応じて、メモリ中のデータを処理する。
- メモリからマス記憶にデータを書き出す。

この操作の例を図5-2に示す。

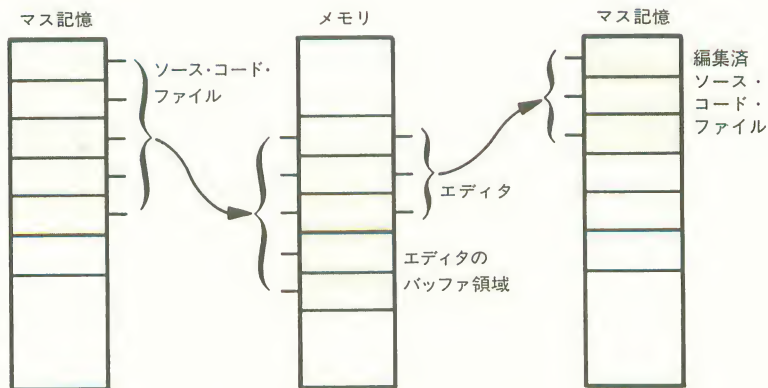


図5-2 基本的なエディタの操作

次の用語は一般に、基本的なタイプのエディタを述べる際に用いられる。

**バッファ**：データはマス記憶から、バッファと呼ばれるメモリ領域に読み込まれる。すべての編集コマンドは、バッファ内のデータを操作する。データは処理が終わると、バッファからマス記憶に書かれる。

**キャラクタまたはラインのポインタ**：エディタは、バッファ内にポインタを保持している。ユーザのコマンドはすべてこのポインタに相対とみなされる。たとえば、4つのキャラクタを削除するコマンドは、ポインタの後の4つのキャラクタを削除する。バッファ内の特定のキャラクタを参照するポインタを用いるエディタもある。このようなエディタは、キャラクタ指向のエディタと呼ばれる。特定の行を参照するポインタを用いるエディタもあり、これはライン指向のエディタと呼ばれる。

### 5.1.1 エディタの機能

エディタはどのような種類の機能を備えるべきであろうか。

エディタは次のような能力を備えている必要がある。

- マス記憶からメモリへのデータのリード
- メモリからマス記憶へのライト
- メモリへのデータの挿入
- メモリからのデータの削除
- バッファ内のキャラクタまたはラインのポインタ位置の変更
- バッファ内容の表示
- 指定ストリングを有するバッファの検索
- バッファ内容の変更
- システム・コマンド

これらの機能を示すために用いるサンプルのエディタは、CRTターミナルで入力されるユーザのコマンドに応答する。コマンドは次の3つのフィールドから成る。

ナンバー      コマンド      ストリング

ナンバーは、特定のコマンドが実行される回数を表わす。このフィールドは10進数として解釈される。このフィールドは省略できる。このフィールドが省略されると、デフォルト値に1が仮定される。

コマンドは、行なわれるべき操作を示す単一のキャラクタである。

ストリングは一連のキャラクタである。実行されるときに、1個あるいはそれ以上のストリングを用いるコマンドがある。このフィールドは省略できる。ストリングは、“#”のキャラクタあるいはリターンのキャラクタで終わる。

すべてのコマンドは、キャリッジ・リターンで終わる。これをⒶで表わす。次はコマンドの例である。

|          |                    |
|----------|--------------------|
| AⒶ       | バッファに1行追加          |
| 10LⒶ     | バッファ内のポインタを10行下に移動 |
| OTHE#ANⒶ | ストリングのTHEをANに変更    |

#### (1) メモリに対するデータの読み込みと書き込み

エディタは、バッファからマス記憶に読み込みあるいは書き込みを行なう能力を備えている必要がある。ユーザは、転送されるデータの量が指定できなければならない。代表的なデータの量には次のものがある。

- 1個または複数のキャラクタ：
- 1行または複数行：たとえば、1行の転送は、キャリッジ・リターンが検出されるまで、すべてのデータを移動する。n行の転送は、n個のキャリッジ・リターンが検出されるまで、すべてのデータを移動する。
- バッファ全体：読み込み操作に対しては、バッファが満たされるまで、マス記憶からバッファへのデータ移動を含む。書き込み操作に対しては、バッファ全体がマス記憶に移動されるまで、バッファからマス記憶へのデータ移動を含む。

有用となる付加的特徴には次のものがある。

- マス記憶からデータを転送して、転送したデータを削除する操作：
- 特定のキャラクタが検出されるまで、データを転送する読み込みあるいは書き込みの

操作：たとえば，ページ単位のデータ転送が可能となる．すなわち，ページ最後のキャラクタ（フォーム・フィード）が検出されるまで，すべての情報を転送する．

例として，バッファに行を付加するサンプルのエディタ・コマンドがAである場合を考える．コマンド

A(r)

はバッファに1行付加する．コマンド

10A(r)

はバッファに10行付加する．コマンド

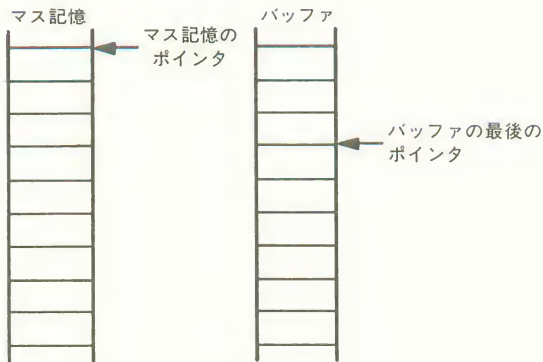
!A(r)

はバッファをマス記憶からの情報で満たす．コマンド

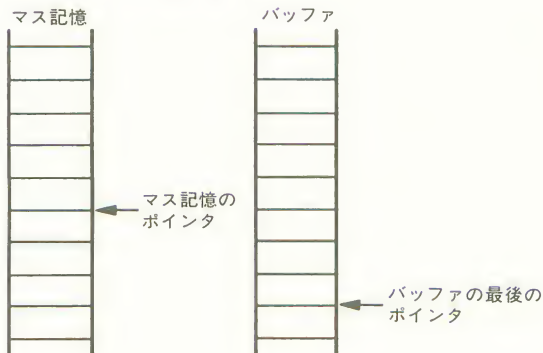
5A

の実行は次のように図示される．

5Aの実行前



5Aの実行後



## (2) バッファへのデータ挿入

エディタは、バッファにデータを付加できる能力を備えていなければならない。ユーザは一般に、次の2つのタイプのソース・コード挿入の1つを行なう必要がある。

- 大量のソース・コードを挿入しなければならない。これは、最初にソース・コードが入力されるとき、あるいはソース・コードの大きい変更を行なうときに生じる。
- 1行または2行のソース・コードを入力しなければならない。これは、デバッグの過程で“バグ”が訂正されるとき、あるいはソース・コードが最初に入力されて、不注意にも1行または2行の入力を忘れていたことが発見されたときに生じる。

多くのエディタは、2つの異なる挿入モードを備えて、これら2つの要求に応じている。すなわち、ユーザが無制限のデータを入力できるモードと、ユーザによって制限された量のデータが入力されるモードがある。

データ挿入のサンプルのエディタ・コマンドがIである場合を考える。次のようなバッファを仮定する。

|       |     |                   |
|-------|-----|-------------------|
|       | MOV | CX, AX            |
| ポインタ→ | ADD | DX, SP            |
|       | JNC | EXIT\$STAGE\$LEFT |

コマンド

|   |     |           |
|---|-----|-----------|
| I | SHR | DX, 1 (I) |
|---|-----|-----------|

が入力されると、バッファは

|     |                   |
|-----|-------------------|
| MOV | CX, AX            |
| ADD | DX, SP            |
| SHR | DX, 1             |
| JNC | EXIT\$STAGE\$LEFT |

に変更される。

## (3) バッファからのデータ削除

エディタは、バッファからデータを除く能力を備えていなければならない。ユーザは、除かれるべきソース・コードの量を指定できる。代表的な量には次のものがある。

- 1個あるいは複数のキャラクタ:
- 1行または複数行: 1行除去は、ライン・ポインタで示される行を除くか、あるいはキャリッジ・リターンが検出されるまで現在のキャラクタ・ポインタで示されるキャラクタで始まるバッファ内のすべてのデータを除く。n行除去は、現在のライン・ポインタ以降のn行、またはn個のキャリッジ・リターンが検出されるまで現在のキャラクタ・ポインタからのすべてのデータを除く。

バッファから行を削除するサンプルのエディタ・コマンドがKである場合を考える。次のようなバッファを仮定する。



```

ポインタ→ MOV    CX, AX
             ADD    DX, SP
             SHR    DX, 1
             JNC    EXIT$STAGE$LEFT

```

コマンド

2K

が入力されると、バッファは

```

MOV    CX, AX
JNC    EXIT$STAGE$LEFT

```

に変更される。

#### (4) キャラクタまたはラインのポインタ移動

エディタは、キャラクタまたはラインのポインタをバッファ内の異なる位置に移動する能力を備えていなければならない。ユーザは、ポインタが移動すべきキャラクタまたはラインの数を指定できる。さらに有用な次の能力を持っている。

- キャラクタまたはラインのポインタの、バッファの先頭への移動。
- キャラクタまたはラインのポインタの、バッファの最後への移動。
- キャラクタまたはラインのポインタの、バッファ内の特定の行への移動。たとえば、キャラクタまたはラインのポインタをバッファの行番号11へ移動することが要求できる。

キャラクタまたはラインのポインタをバッファ内で上下に移動するためのサンプルのエディタ・コマンドがLである場合を考える。次のようなバッファを仮定する。

```

ポインタ→ MOV    CX, AX
             ADD    DX, SP
             SHR    DX, 1
             JNC    EXIT$STAGE$LEFT
             TEST   BX, 40H
             JZ     DONT$MESS$WITH$BILL

```

コマンド

4L

が入力されると、バッファは変化しないが、ポインタは JZ DONT\$MESS\$WITH\$BILL の命令を示す。さらにコマンド

-3L

により、バッファはやはり変化しないが、ポインタは SHR DX, 1 の命令を示す。

#### (5) バッファ内容の表示

エディタは、バッファをユーザのターミナルに表示する能力を備えていなければならない。ユーザは、表示されるキャラクタや行の数を指定することができる。さらに有用な能力に次のものがある。

- CRTターミナルがあれば、スクリーン全体にデータを自動的に表示するのに便利である。さらに、一度に1行あるいは画面全体に表示する、バッファによるスクロール



の能力もさらに便利である。

バッファから行をC R Tターミナルに表示するサンプルのエディタ・コマンドがTである場合を考える。バッファ内容が

|       |     |                                |
|-------|-----|--------------------------------|
|       | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
| ポインタ→ | CMP | AL, COLUMN\$4\$DIGIT           |
|       | JNZ | TOUCH\$TONE\$ENCODE            |
|       | MOV | [DI], MESSAGE\$STARTED\$CODE   |

で、コマンド

2T

を入力すると、

|  |     |                      |
|--|-----|----------------------|
|  | CMP | AL, COLUMN\$4\$DIGIT |
|  | JNZ | TOUCH\$TONE\$ENCODE  |

の行がC R Tターミナルに表示される。

#### (6) スtringのバッファ検索

エディタは、ユーザ指定のキャラクタのStringのバッファ検索の能力を備えている必要がある。非常に有用な付加的能力に、特定のStringを含むすべてのソース・コードの検索がある。たとえば、ソース・コードが最初に入力されたとき、入力間違いがしばしば起こる。検索機能を変更機能と共に用いることによって、ソース・コードを最小の労力で修正することができる。

バッファ検索のサンプルのエディタ・コマンドがSである場合を考える。もし、バッファが

|       |     |                                |
|-------|-----|--------------------------------|
|       | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
| ポインタ→ | CMP | AL, COLUMN\$4\$DIGIT           |
|       | JNZ | TOUCH\$TONE\$ENCODE            |
|       | MOV | [DI], MESSAGE\$STARTED\$CODE   |

で、コマンド

STONE①

を入力すると、その結果は、エディタでキャラクタ・ポインタあるいはライン・ポインタのどちらを採用しているかに依存する。ライン・ポインタの場合、バッファは変化しない。しかし、ライン・ポインタの位置は次のように変更される。

|       |     |                                |
|-------|-----|--------------------------------|
|       | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
|       | CMP | AL, COLUMN\$4\$DIGIT           |
| ポインタ→ | JNZ | TOUCH\$TONE\$ENCODE            |
|       | MOV | [DI], MESSAGE\$STARTED\$CODE   |

キャラクタ・ポインタの場合は、ポインタは次のように変更される。

|       |     |                                |
|-------|-----|--------------------------------|
|       | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
| ポインタ→ | CMP | AL, COLUMN\$4\$DIGIT           |
|       | JNZ | TOUCH\$TONE\$ENCODE            |
|       | MOV | [DI], MESSAGE\$STARTED\$CODE   |

## (7) バッファのストリングの変更

エディタは、バッファ内のデータを変更する能力を備えていなければならない。ユーザは、バッファに存在する任意のストリングをユーザ指定のストリングで置き換えることを指定できる。削除機能は、ユーザ指定のストリングがない、変更機能の縮小された場合と考えることができる。

バッファのデータを変更するサンプルのエディタ・コマンドが C ストリング 1 # ストリング 2 である場合を考える。ここでコマンドは、次にバッファ内でストリング 1 の位置する場所を探して、それをストリング 2 で置き換えて、機能を果たす。バッファが次のようであるとする。

|        |     |                                |
|--------|-----|--------------------------------|
|        | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
| ポインタ → | CMP | AL, COLUMN\$4\$DIGIT           |
|        | JNZ | TOUCH\$TONE\$ENCODE            |
|        | MOV | [DI], MESSAGE\$STARTED\$CODE   |

コマンド

CCODE#TRANCE①

を入力すると、バッファは次のように変更される。

|        |     |                                |
|--------|-----|--------------------------------|
|        | IN  | AL, TOUCH\$TONE\$DECODER\$PORT |
|        | CMP | AL, COLUMN\$4\$DIGIT           |
| ポインタ → | JNZ | TOUCH\$TONE\$ENTRANCE          |
|        | MOV | [DI], MESSAGE\$STARTED\$CODE   |

### 5.1.2 システム・コマンド

エディタは、ユーザが合理的な方法で編集を終了させられるコマンドを備えていなければならない。合理的終了方法には次のものがある。

- バッファ内のすべてのデータをマス記憶へ移動する。
- バッファを通してすべての未処理のソース・コードをマス記憶に移動する。これは標準的な終了方法と考えられる。
- バッファをクリアせずに直ちに編集を終える。この方法は、分別が足らないか運の悪いユーザの操作がソース・コードに破滅的な影響を与えた場合に用いられる。マス記憶の形式によっては、ソース・コードを原形に復元することができる。

上のタイプのコマンドは、基本的なエディタにとって必要な要素である。より高度なエディタは次のような能力を持つ。

1. 個々のコマンドを、コマンド・ストリングに連結する：
2. コマンド・ストリングの多重繰り返し：たとえば、これはソース・コード中の特定のストリングすべてを変更する場合に特に有用である。
3. より高度なファイル処理：この解説では、ファイルの概念は避けている。ソース・コードは、マス記憶に存在する点から論じているだけである。より進んだエディタは、

たとえばハード・ディスクなどの高速マス記憶から一般に起動され、有力なデータ・ファイル操作能力を備えているオペレーティング・システムとのインターフェイスを持つ。実際、バッファからの読み込みと書き込みのユーザの責任を軽減するエディタがある。このエディタでは、ソース・コードの読み込みや書き込みについて考えることなく、ソース・コード全体をスクロールさせることができる。この機能はエディタによって自動的に行なわれる。

4. 算術演算能力：非常に有力な計算器として用いられるエディタもある。
5. バッファのある部分を抜き出して、後で用いるために保存する能力：この能力は、ソース・コードの再配列が必要なとき、たとえば、ファイルの先頭の100行のソース・コードをファイルの中央に移動しなければならないとき、非常に有用となる。
6. スtring操作に“あいまいな”要素を含む能力：たとえば、第1のキャラクタがAで、最後の3つのキャラクタがCDEで、第2のキャラクタがあいまいな、すなわち任意のキャラクタの、5個のキャラクタのStringを検索するために、検索操作にはA\*CDEを用いる。

## 5.2 アセンブラ

多くのアセンブラは次の機能を行なう。

- アセンブリ言語のソース・コードを別々のステートメントに分割する：
- アセンブリ言語のステートメントを構成部分に分ける：この部分には、ラベル、アセンブリ言語のオペレータ、アセンブラ命令、アセンブリ言語のオペレータのためのオペランド、さらに注釈が含まれる。
- アセンブリ言語の規則に従って、構成部分を処理する：この処理から、アセンブラはオブジェクト・コード・ファイルとシンボル・テーブルを生成する。
- ファイルをマス記憶に書き込む：このファイルには、オブジェクト・コード・ファイル、リスティング・ファイル（これはオブジェクト・コード・ファイルとソース・コード・ファイルから成る）、さらにシンボル・テーブル・ファイルが含まれる。

ソース・コードをアセンブリ言語のステートメントに分割することは、ほとんどのソース・コード・ファイルは1行が1つのステートメントで構成されている、すなわち、2つのキャリッジ・リターンの間には1つのステートメントが存在するので、かなり容易な作業である。1行に1つ以上のステートメントが可能なアセンブラもある。このようなステートメントは通常、キャリッジ・リターンと同様に取り扱われる特殊な区切りキャラクタで分離されている。

アセンブラを非常に有用な道具にする機能は、アセンブリ言語のステートメントの処理にある。アセンブリ言語のステートメントは次のような部分を持つ。

- ラベル：アセンブリ言語の命令には、ラベルがある場合とない場合がある。ラベルが存在すれば、ロケーション・カウンタの現在の値と共にシンボル・テーブルに記憶される。より複雑なアセンブラでは、オペレータのタイプあるいは命令によって、より

多くの情報が記憶される。

- オペレータ：オペレータは、ADC, STD, INなどのアセンブリ言語のニーモニックかあるいはアセンブラ命令である。アセンブリ言語のニーモニックは、アセンブラによってオブジェクト・コードに翻訳される。たとえば、ADC命令は数100の異なるオブジェクト・コードを生成するが、ADC AX, DXの命令は唯一のオブジェクト・コードを生成する。

アセンブラ命令は、アセンブラがオブジェクト・コード・ファイルとリスティング・ファイルを生成するときに用いる各種の機能を制御するために用いられる。これは次のような制御を行なう。

- ソース・コードがアセンブルされるロケーション：プログラム・メモリのロケーションに対して絶対アドレスを含むプログラムは、それがメモリのどこに位置するかを知る必要がある。サンプルのアセンブラのロケーション指定の命令が、ORGである場合を考える。アセンブラ命令

ORG 0400H

がソース・コードに含まれていれば、これに続くアセンブリ言語のステートメントは、プログラム・カウンタが0400Hに設定されているとしてアセンブルされる。

- プログラムの実行開始アドレス：これは通常、オブジェクト・コード内に記憶される。ほとんどのアセンブラでは、プログラムの開始アドレスを、ソース・コードの最後のステートメント、ENDステートメントで指定することができる。ソース・アセンブラが開始アドレスを指定するためにENDステートメントを用いていると仮定する。

END START\$OF\$PROGRAM

のステートメントが、ソース・コードの最後のステートメントならば、アセンブラは、START\$OF\$PROGRAMのアドレスを開始アドレスとして含むオブジェクト・コードを生成する。

- リスティング・ファイルの形式：リスティング・ファイルの形式を制御する命令は、ページ数を付けたり、ソースとオブジェクトのコードのある部分がリスティング・ファイルに含まれるかどうかを決めたり、リスティング・ファイルに関連した見出しなどの制御を行なう。
- データ・メモリのロケーションの初期値：
- オペランド：オペランドを必要とするアセンブリ言語のニーモニックと命令に対して、オペランドは次のように種々の異なる形で用いられる。

レジスタ名

数値（いくつかの異なるベースの1つで）

変数名

ラベル

ASCIIキャラクタのストリング式（算術演算あるいは論理演算のオペレータによる上記の組み合わせ）

- 注釈：注釈は、プログラムの動作の説明に用いられる。これは、アセンブラでは無視-



されるが、プログラムの修正に興味を持つユーザには必要である。

アセンブラが行なう翻訳過程は、かなり簡単な作業である。ステートメントが構成部分に分割されると、構成部分からテーブルを用いてオブジェクト・コードの部分を作る。これを次に最終的なオブジェクト・コードに組み立てる。

### 5.3 デバッガ

デバッガは、オブジェクト・プログラムからエラーを取り除く際の援助に用いられる開発の道具である。デバッガは、エディタとアセンブラと同様に、複雑に異なる。最も基本的なデバッガは、ユーザに次のことを可能とする要素を含む。

- 実行の制御
- レジスタまたはメモリの表示

デバッガでは、次のような機能を用いて実行を制御できる。

- シングル・ステップの機能：シングル・ステップの機能で、オブジェクト・コードを一度に1命令実行することができる。ユーザは各命令実行の間のレジスタやメモリを調べることができる。このことは有望にも、ユーザに命令が要求される機能を果たしていることを保証するのに十分である。より複雑なデバッガは、実行される命令の正確な数の指定や各命令の実行に続いて表示されるレジスタ、あるいはメモリの指定ができる、シングル・ステップ・ルーチンの高度な形式を有している。
- ブレークポイントの機能：ブレークポイントの機能により、ユーザの指定した位置のオブジェクト・コードに、特殊なコード、8086の場合はソフトウェア・インタラプト命令を置くことによって、実行を制御することができる。特殊なコードが実行されると、その結果、制御がデバッガに移り、ユーザのオブジェクト・コードの実行が停止させられる。この時点で、デバッガは、特殊なコードに変更されていた位置に元のオブジェクト・コードを戻して、CPUの状態が調べられるようにする。

デバッガは一般に、メモリの任意の部分とCPUの内部レジスタの内容を表示することができ、したがって状態を完全に調べられる。

より複雑なデバッガでは、次のことが可能である。

- メモリやレジスタの内容の変更
- オブジェクト・コードの実行のトレース
- オブジェクト・コードのアセンブルやディスアセンブル
- マス記憶からの読み込みや書き込み
- 簡単な算術演算機能の実行
- より高度なブレークポイントの実行
- シンボル・テーブルの操作

代表的なデバッガは、次のようなメモリやレジスタの内容変更に対して、いくつかの機能を備えている。

- メモリを調べて必要ならば変更する。

- 一連のメモリを定数で満たす。
- メモリ・ブロックの内容を他のメモリ・ブロックに移動する。

メモリを定数で満たすサンプルのデバッグ・コマンドが F addr<sub>1</sub>, addr<sub>2</sub>, 定数 である場合を考える。このコマンドは、addr<sub>1</sub> から addr<sub>2</sub> (このアドレスも含む) までのメモリのすべてを定数で満たす。たとえば、デバッグ・コマンド

F100, 17F, 20

が入力されると、デバッグは 100<sub>16</sub> から 17F<sub>16</sub> のすべての位置に定数 20<sub>16</sub> を書き込む。

オブジェクト・コード実行のトレースを可能とするデバッグは、一般にこの機能をシングル・ステップの拡張として行なう。ユーザは通常、実行されるべきステップ数と表示されるべき情報の種類を指定して、これによってプログラムの実行を見る (トレースする) ことができる。

16進ではなくて、アセンブリ言語の命令としてメモリ内容の表示を指定できるような、メモリ表示のより高度な形式を備えているデバッグもある。たとえば

D 400, 405

では

400 E4 10 24 40 74 FA

となるが、この代わりに

L 400

のコマンドは、メモリ 400<sub>16</sub> - 405<sub>16</sub> を次のように表示する。

400 IN AL, 10  
402 AND AL, 40  
404 JZ 400

さらに、メモリ内容の変更に用いられる基本的なアセンブラの機能を備えているデバッグがある。次のオブジェクト・コードを代入する代わりに、

S404 75,  
405 FA

ソース・コードの代入ができる。

A404 JNZ  
400

デバッグはしばしば、マス記憶に対してデータの読み込みと書き込みを行なう基本的能力を備えている。代表的能力には次のものがある。

- ユーザ指定のメモリ領域にマス記憶からオブジェクト・コード・ファイルを読み出す。
- ユーザ指定のメモリ領域からマス記憶にオブジェクト・コード・ファイルを書き込む。
- 基本的なペーパー・テープ処理の機能。

多くのデバッグは、16進の算術演算機能を備えている。ユーザは通常、2つの16進数値を入力して、デバッグに2つの数値の和と差を計算して表示させることができる。

より高度なブレークポイントを持つデバッグは、一般に次のものを備えている。

- 各ブレークポイントのパス・カウント：ブレークポイント・アドレスから命令がフェ



ッチされる度に、パス・カウントは減少させられる。パス・カウントが0になれば、ユーザのプログラムは一時中止させられて、制御はデバッガに戻る。そしてユーザはCPUの状態を見ることができる。この特徴は、プログラムのループをデバッグするときに特に有用となる。たとえば、53回目のループに何か問題が生じていると考えられるならば、53のパス・カウントでループ中にブレークポイントを設定して、妨げとなる繰り返しを都合のよいときに調べることができる。もしパス・カウントの機能が利用できなければ、53回の繰り返しの間にユーザのプログラム実行を停止させることは容易ではない。

- メモリがデータのためにアクセスされたときにユーザのオブジェクト・コードの実行を一時中止させる、ハードウェア・ブレークポイントの機能。これは、メモリが予測できないように壊されている場合に非常に有用となる。メモリ・アクセスのブレークにより、一般に問題の原因の識別が可能となる。

高度なデバッガは、アセンブラによって作られたシンボル・テーブルを用いて動作する。このデバッガでは、名前でメモリを参照することができる。この特徴は、リロケートブル・オブジェクト・コードで動作するときに非常に役立つ。マップとリストを用いて、特定の変数のアドレスを計算する代わりに、変数は名前で直接に参照することができる。

# 第6章

## 8086アセンブリ言語の プログラミング例

この章では、8086アセンブリ言語のプログラミングの2つの例、ソート・プログラムとI/Oドライバについて解説する。これら2つの例の仕様とプログラム設計の労苦は、既に2章に示されている。

### 6.1 ソート・プログラム

ソート・プログラムは、次の分離した3つのモジュールに分かれる。

- テープの読み込み
- レコードのソート
- テープの書き込み

“テープの読み込み”は次の1つのサブルーチンをコールしている。

- テープからのレコードの読み込み

“レコードのソート”は次の4つのサブルーチンをコールしている。

- サブソートを一時記憶領域へ移動
- キーの比較
- ポインタの計算
- レコードの移動

“テープの書き込み”は次の1つのサブルーチンをコールしている。

- テープへのレコードの書き込み

ソース・コードを見ると、サブルーチンの呼び出しのすべてが必要ではないことが明らかとなる。たとえば、“テープからのレコードの読み込み”ルーチンは、ソース・コードのただ1つのステートメントによってだけ呼び出されている。しかし、“テープからのレコードの読み込み”ルーチン全体が、それが呼び出されている位置のソース・コードに含まれている場合よりも、ソース・コードのモジュールはすっきりしている。

ソート・プログラムはアブノーマルの終了を2つ有している。この2つの終了は共に、テープ・コントローラの読み込み / 書き込み・ルーチン中で発生する。このプログラムのためには、オペレーティング・システムの存在と、もし異常が検出されれば、オペレーティング・システムが適当な装置にエラー・メッセージを出力することが假定されている。

|                                                          |      |                        |                            |
|----------------------------------------------------------|------|------------------------|----------------------------|
| ;EQUATES FOR SORT ROUTINE                                |      |                        |                            |
| TAPE\$COMMAND\$PORT                                      | EQU  | 20H                    | ;ARBITRARY #'S. TYPICALLY  |
| TAPE\$STATUS\$PORT                                       | EQU  | 20H                    | ;THESE #'S WOULD BE LISTED |
| TAPE\$DATA\$PORT                                         | EQU  | 22H                    | ;IN THE SPECIFICATION.     |
| READ\$TAPE\$COMMAND                                      | EQU  | 01H                    | ;FROM SPECIFICATION        |
| WRITE\$TAPE\$COMMAND                                     | EQU  | 02H                    |                            |
| OPERATION\$COMPLETE\$FLAG                                | EQU  | 04H                    |                            |
| TAPE\$ERROR\$STATUS                                      | EQU  | 080H                   |                            |
| TAPE\$ERROR\$FLAG                                        | EQU  | 044H                   | ;USED BY SYSTEM            |
| ;EXTERNAL REFERENCES                                     |      |                        |                            |
| EXTRN SYSTEM:                                            | FAR  |                        |                            |
| EXTRN SYSTEM\$ERROR                                      | FAR  |                        |                            |
| DATA SEGMENT                                             |      |                        |                            |
| ;RAM LOCATIONS FOR SORT PROGRAM                          |      |                        |                            |
| RECORD\$TEMP                                             | DB   | 2                      |                            |
| KEY\$TEMP                                                | DB   | 10                     |                            |
| INDEX                                                    | DW   | 1                      |                            |
| INTERVAL                                                 | DW   | 1                      |                            |
| SUBSORT\$COUNTER                                         | DW   | 1                      |                            |
| RECORD\$COUNT                                            | DW   | 1                      |                            |
| TAPE\$BUFFER                                             | DB   | 140 DUP(?)             |                            |
| SORT\$AREA                                               | DB   | 4000 DUP (12 DUP(0))   |                            |
| DATA ENDS                                                |      |                        |                            |
| CODE SEGMENT                                             |      |                        |                            |
| ASSUME CS: CODE, DS: DATA, ES: DATA                      |      |                        |                            |
| MAIN                                                     |      |                        |                            |
|                                                          | MOV  | AX,DATA                | ;LOAD SEGMENT REGISTERS    |
|                                                          | MOV  | DS,AX                  |                            |
|                                                          | MOV  | ES,AX                  |                            |
|                                                          | MOV  | RECORD\$COUNT,0        | ;SET # OF RECORDS TO 0     |
|                                                          | MOV  | DI,OFFSET SORT\$AREA   | ;POINT DI AT SORT AREA     |
| ; READ THE TAPE MODULE OPERATES BY                       |      |                        |                            |
| ; 1 READING A TAPE RECORD                                |      |                        |                            |
| ; 2 CHECKING FOR DONE                                    |      |                        |                            |
| ; 3 MOVING 6 WORDS FROM THE TAPE BUFFER TO THE SORT AREA |      |                        |                            |
| ; 4 UPDATING THE # OF RECORDS                            |      |                        |                            |
| ;                                                        |      |                        |                            |
| ;                                                        |      |                        |                            |
| READ\$THE\$TAPE                                          | CALL | READ\$TAPE\$BUFFER     | ;READ 128 BYTES            |
|                                                          | MOV  | SI,OFFSET TAPE\$BUFFER | ;TEST FOR EOF RECORD       |
|                                                          | CMP  | [SI],OFFFH             | ;GO SORT IF EOF            |
|                                                          | JZ   | SORT                   | ;NOT EOF, MOVE RECORD #    |
|                                                          | MOV  | CX,12                  | ;AND KEY                   |
|                                                          | REP  | MOVS TAPE\$BUFFER,     | ;INCREMENT # OF RECORDS    |
|                                                          |      | SORT\$AREA             |                            |
|                                                          | INC  | RECORD\$COUNT          | ;GET ANOTHER BYTE          |
|                                                          | JMP  | READ\$THE\$TAPE        |                            |

ステートメント

```
MOV     SI,OFFSET TAPE$BUFFER      ;TEST FOR EOF RECORD
```

のOFFSETオペレータは、イミディエイト・データとしてSIレジスタにTAPE\$BUFFERのアドレスをロードするオブジェクト・コードの生成に用いられる。

ステートメント

```
MOV     SI,TAPE$BUFFER
```

は、TAPE\$BUFFERの内容をSIレジスタにロードするオブジェクト・コードを生成することに注意。OFFSETオペレータは標準のインテル8086アセンブラの特徴であり、8086マイクロプロセッサの特徴ではない。

奇数バイトの移動よりも、偶数バイトの移動が容易であることに注意。奇数バイトを移動するためには、2つの方法が考えられる。その1つを次に示す。

```
MOV     CX,ODD$NUMBER              ;LOAD # OF BYTES
REP     MOVSB                      ;TO MOVE
```

この方法では、偶数バイトの移動に用いられているのと同数バイトのオブジェクト・コードを用いているが、実行には2倍の時間を要する。もう1つの方法を次に示す。

```
MOV     CX,ODD$NUMBER              ;LOAD # OF WORDS TO MOVE
SHR     CX,1
REP     MOVSW
MOVSB                                ;MOVE LAST BYTE
```

この方法は、さらに1バイトのオブジェクト・コードを必要とするが、偶数バイト移動ルーチンと同じ時間で実行される。

THE SORT MODULE IS A STRAIGHTFORWARD RENDITION OF THE ALGORITHM  
PRESENTED IN CHAPTER 3

|                         |                                 |                             |
|-------------------------|---------------------------------|-----------------------------|
| START:                  | MOV AX,RECORD\$COUNT            | ;INITIALIZE INTERVAL TO     |
|                         | MOV INTERVAL,AX                 | ;RECORD COUNT               |
| NEW\$INTERVAL:          | SHR INTERVAL,1                  | ;DIVIDE INTERVAL BY 2       |
|                         | JZ WRITE\$TO\$TAPE              |                             |
|                         | MOV AX,RECORD\$COUNT            | ;SUBSORT CTR=RECORD\$       |
|                         |                                 | COUNT-INTERVAL              |
|                         | SUB AX,INTERVAL                 |                             |
|                         | MOV SUBSORT\$COUNTER,AX         |                             |
| NEXT\$SUBSORT\$COUNTER: | INC SUBSORT\$COUNTER            |                             |
|                         | MOV AX,SUBSORT\$COUNTER         |                             |
|                         | CMP AX,RECORD\$COUNT            | ;TEST FOR NEW INTERVAL      |
|                         | JG NEW\$INTERVAL                |                             |
|                         | CALL MOVE\$SUBSORT\$ TO\$TEMP   | ;SAVE CURRENT RECORD        |
|                         | MOV AX,SUBSORT\$COUNTER         | ;INDEX=SUBSORT CTR-INTERVAL |
|                         | SUB AX,INTERVAL                 |                             |
| TEST\$KEYS:             | MOV INDEX,AX                    |                             |
|                         | CALL COMPARE\$KEYS              |                             |
|                         | JGE FOUND\$THIS\$RECORDS \$SPOT |                             |
|                         | MOV AX,INDEX                    |                             |
|                         | CALL COMPUTE\$POINTER           |                             |
|                         | MOV SI,AX                       |                             |
|                         | CALL MOVE\$RECORD               |                             |

```

MOV AX,INTERVAL ,INDEX-INTERVAL=INDEX
SUB INDEX,AX
JGE TEST$KEYS
FOUND$THIS$RECORDS$SPOT: MOV SI,OFFSET RECORD$TEMP
CALL MOVE$RECORD
JMP NEXT$SUBSORT$COUNTER

```

命令

```
SHR INTERVAL,1
```

は、

```

MOV AX,INTERVAL
SHR AX,1
MOV INTERVAL,AX

```

よりも、メモリ使用と時間消費の両方からより効率的であることに注意。すべての場合、情報をレジスタに移動し、それを操作して結果をメモリに戻すよりも、直接にメモリで処理する方がより能率がよい。

```

; WRITE TAPE OPERATES BY
; 1. INITIALIZING PTRS TO THE TAPE BUFFER AND SORT AREA
; 2. MOVING 12 BYTES AT A TIME UNTIL EITHER
;
;

```

```

; 128 BYTES HAVE BEEN MOVED
; END OF FILE IS REACHED
;
;

```

```

; 3 IF 128 BYTES, WRITE A TAPE RECORD
; 4 IF END OF FILE, APPEND AN EOF RECORD, THEN WRITE
; THE LAST TAPE RECORD
;

```

```

WRITE$TO$TAPE: MOV SI,OFFSET SORT$AREA
NEXT$TAPE$BUFFER: MOV DI,OFFSET TAPE$BUFFER
MOVE$NEXT$RECORD: MOV CX,12 ,GET READY TO MOVE 12 BYTES
REP MOVSB TAPE$BUFFER,SORT$AREA

CMP DI,OFFSET TAPE$BUFFER + 128
JL UPDATE$RECORD$COUNT ,TEST FOR MOVED FULL BUFFER

PUSH SI ,SAVE POINTERS
PUSH DI
CALL WRITE$TAPE$BUFFER ,WRITE 128 BYTES TO TAPE
POP DI ,RESTORE POINTERS
POP SI
MOV AX,OFFSET TAPE$BUFFER + 128 ,ANY EXTRAS IN END OF TAPE
SUB DI,AX ,BUFFER

```

```

; NOTE: TO FILL 128 BYTES REQUIRES MOVING 11 RECORDS OR 11 X 12 =
; 132 BYTES INTO TAPE BUFFER

```

```

MOV CX,DI ,CX GETS COUNT
MOV DI,OFFSET TAPE$BUFFER
JZ UPDATE$RECORD$COUNT ,JUMP IF NO EXTRAS
PUSH SI ,SAVE POINTER INTO SORT AREA
MOV SI,AX
REP MOVSB TAPE$BUFFER,TAPE$BUFFER ,MOVE EXTRAS DOWN TO START
POP SI ,OF TAPE BUFFER
UPDATE$RECORD$COUNT: DEC RECORD$COUNT
JNZ MOVE$NEXT$RECORD

```



|            |      |                               |                             |
|------------|------|-------------------------------|-----------------------------|
|            | CMP  | DI,OFFSET TAPE\$BUFFER        | ;TEST IF ONE MORE RECORD    |
|            | JZ   | WRITE\$EOF                    | ;MUST BE WRITTEN BEFORE EOF |
|            | MOV  | CX, OFFSET TAPE\$BUFFER + 128 |                             |
|            | SUB  | CX,DI                         | ;ZERO OUT THE REST OF       |
|            | XOR  | AL,AL                         | ;THE TAPE BUFFER            |
|            | REP  | STOS TAPE\$BUFFER             |                             |
|            | CALL | WRITE\$TAPE\$BUFFER           | ;WRITE LAST TAPE RECORD     |
| WRITE\$EOF | MOV  | TAPE\$BUFFER,OFFFH            | ;MOVE IN END OF FILE RECORD |
|            | CALL | WRITE\$TAPE\$BUFFER           | ;WRITE EOF RECORD, A        |
|            | JMP  | SYSTEM                        | ;RECORD WITH FFFF IN        |
|            |      |                               | ;THE FIRST TWO BYTES        |
|            |      |                               | ;END OF PROGRAM             |
|            |      |                               | ;RETURN TO SYSTEM           |

## ;PROCEDURES CALLED BY MAIN PROGRAM

|                      |      |                      |                        |
|----------------------|------|----------------------|------------------------|
| COMPUTE\$POINTER     | PROC | NEAR                 |                        |
| AX HAS INDEX         |      |                      |                        |
| RETURN ADDR IS IN AX |      |                      |                        |
| DX IS NOW 0          |      |                      |                        |
|                      | MOV  | CX,12                |                        |
|                      | MUL  | CX                   |                        |
|                      | ADD  | AX,OFFSET SORT\$AREA | ;ADD ADDRESS, NOT DATA |
|                      | RET  |                      |                        |
| COMPUTE\$POINTER     | ENDP |                      |                        |

このモジュールは、

|     |       |
|-----|-------|
| MOV | CX,12 |
| MUL | CX    |

を

|     |       |
|-----|-------|
| SHL | AX,1  |
| SHL | AX,1  |
| MOV | CX,AX |
| SHL | AX,1  |
| ADD | AX,CX |

で置き換えることによって処理速度を上げることができる。

MOV/MULの命令は、実行に126サイクルを要する。2番目の命令は実行に11サイクルを要し、しかもDXレジスタを壊さない。ただし、MOV/MULの命令ではプログラム・メモリの5バイトだけでよいのに対して、2番目の命令は10バイトを要する。

|                                                   |      |                         |                             |
|---------------------------------------------------|------|-------------------------|-----------------------------|
| ; WRITE TAPE BUFFER OPERATES BY                   |      |                         |                             |
| ; 1. POINTING AT THE TAPE BUFFER                  |      |                         |                             |
| ; 2. INITIALIZING THE TAPE CONTROLLER FOR A WRITE |      |                         |                             |
| ; 3. CHECKING FOR STATUS ERRORS                   |      |                         |                             |
| ; 4. CHECKING FOR OPERATION DONE                  |      |                         |                             |
| ; 5. WRITING TO THE TAPE DATA PORT                |      |                         |                             |
| WRITE\$TAPE\$BUFFER                               | PROC | NEAR                    |                             |
|                                                   | MOV  | SI,OFFSET TAPE\$BUFFER  | ;GET ADDRESS OF TAPE BUFFER |
|                                                   | MOV  | AL,WRITE\$TAPE\$COMMAND | ;START TAPE WRITE           |
|                                                   | OUT  | TAPE\$COMMAND\$PORT,AL  |                             |
| GET\$TAPE\$STATUS:                                | IN   | AL,TAPE\$STATUS\$PORT   | ;CHECK FOR ERRORS           |
|                                                   | TEST | AL,TAPE\$ERROR\$STATUS  |                             |



```

                                JNZ    OUTPUT$TAPE$ERROR
                                TEST   AL,OPERATION$COMPLETE$FLAG ;TEST FOR DONE
                                JNZ    WRITE$COMPLETE
                                LODSB
                                OUT     TAPE$DATA$PORT,AL           ;GET A BYTE
                                JMP     GET$TAPE$STATUS             ;SHIP IT OUT
OUTPUT$TAPE$ERROR: MOV     AH,TAPE$ERROR$FLAG
                                JMP     SYSTEM$ERROR
WRITE$COMPLETE: RET
WRITE$TAPE$BUFFER ENDP

```

### TEST の操作

```
TEST    AL,TAPE$ERROR$STATUS
```

は、ALレジスタのステータス・バイトを次の操作のために保存するために、AND操作の代わりに用いられる。

ALレジスタの内容は、調べる操作が終了した後では意味を持たないので、

```
TEST    AL,OPERATION$COMPLETE$FLAG
```

の操作は

```
AND     AL,OPERATION$COMPLETE$FLAG
```

で置き換えることができる。

```

; READ TAPE BUFFER OPERATES BY
; 1. INITIALIZING TAPE CONTROLLER TO READ
; 2. CHECKING FOR TAPE ERRORS
; 3. CHECKING FOR COMPLETION
; 4. READING DATA FROM TAPE DATA PORT
;
; THIS ROUTINE USES SI AND AL
;
; IF AN ERROR OCCURS, THIS ROUTINE BRANCHES TO THE SYSTEM
;
;
READ$TAPE$BUFFER  PROC    NEAR
MOV     SI,OFFSET TAPE$BUFFER           ;POINT AT TAPE BUFFER
MOV     AL,READ$TAPE$COMMAND             ;TELL TAPE TO READ
OUT     TAPE$COMMAND$PORT,AL
GET$STATUS: IN     AL,TAPE$STATUS$PORT
TEST    AL,TAPE$ERROR$STATUS             ;CHECK FOR TAPE ERRORS
JNZ     TAPE$ERROR
TEST    AL,OPERATION$COMPLETE$FLAG       ;CHECK FOR DONE
JNZ     READ$COMPLETE
IN      AL,TAPE$DATA$PORT                 ;GET DATA
MOV     [SI],AL                           ;SAVE DATA
INC     SI
JMP     GET$STATUS
TAPE$ERROR: MOV     AH,TAPE$ERROR$FLAG     ;CALL SYSTEM
JMP     SYSTEM$ERROR                     ;ERROR PROCESSOR
READ$COMPLETE: RET
READ$TAPE$BUFFER ENDP

```

このルーチンとWRITE\$TAPE\$BUFFER ルーチンとの類似性に注意。読み込みと書き込みのルーチンでは、2つの異なる点がある。

```
WRITE$TAPE$BUFFER    MOV    AL,WRITE$TAPE$COMMAND
```

は

```
READ$TAPE$BUFFER    MOV    AL,READ$TAPE$COMMAND
```

で置き換えられ、

```
                      LODSB
                      OUT     AL,TAPE$DATA$PORT
```

は

```
                      IN      AL,TAPE$DATA$PORT
                      MOV     [SI],AL
                      INC      SI
```

で置き換えられている。

読者は練習として、テープ・バッファの読み込みと書き込みのルーチンが共通コードを共有するように、この2つのルーチンを1つにしてみよ。テープ・バッファのポインタとしてDIレジスタを用いればより効果的となることに注意。たとえば、

```
MOV    [SI],AL
INC     SI
```

は

```
STOSB
```

で置き換えられる。ただし、DIレジスタはメインのREAD\$THE\$TAPE モジュールで用いられている。

```
; COMPARE KEYS OPERATES BY COMPARING KEY (INDEX) WITH KEYTEMP
; 1. CALCULATE INDEX
; 2. COMPARE KEYS UNTIL
;   * DIFFERENCE IS FOUND
;   * 10 BYTES HAVE BEEN COMPARED

COMPARE$KEYS    PROC    NEAR
MOV             AX,INDEX                ;GET INDEX
CALL            COMPUTE$POINTER
INC             AX                      ;POINT PAST RECORD #
INC             AX
MOV             DI,AX                  ;MOVE TO DI FOR COMPARE
MOV             SI,OFFSET KEY$TEMP
MOV             CX,0010                ;10 BYTES TO COMPARE
CMPS            KEY$TEMP, SORT$AREA
;COMPARE 5 WORDS

RET
COMPARE$KEYS    ENDP
```

命令

```
MOV     SI,OFFSET KEY$TEMP
```

は、KEY\$TEMP の値ではなく、KEY\$TEMP のアドレスをSIレジスタにロードする。

```

; MOVE RECORD OPERATES BY MOVING WHATEVER SI POINTS AT TO THE LOCATIONS
; POINTED TO BY INDEX INTERVAL
; 1. CALCULATE PTR. FOR INDEX + INTERVAL
; 2. MOVE 12 BYTES

MOVE$RECORD      PROC    NEAR
MOV              AX,INDEX
ADD              AX,INTERVAL          ;CALC INDEX + INCREMENT
CALL            COMPUTE$POINTER
MOV              DI,AX
REP              MOVS SORT$AREA,SORT$AREA ;COMPUTER POINTER
                                           RETURNS CX=12
RET
MOVE$RECORD      ENDP

```

メモリ領域に対して時間の節約が必要ならば、

```

MOV      DI,AX
REP      MOVSB
RET

```

の命令は

```

MOV      DI,AX
SHR      CX,1
REP      MOVSW
RET

```

で置き換えられる。

MOVSBをMOVSWで置き換えることにより、 $6 \times 17 = 102$  クロックが節約される。シフト命令に必要な2サイクルを減じると、正味100サイクルの節約が実現される。

オペランドの指定が、そのオペランドのタイプからバイトあるいはワードの操作を決定するアセンブラがあることに注意。この例では、

```
MOVSB  SORT$AREA, SORT$AREA
```

は、SORT\$AREAがバイト構成のバッファなので、バイト操作となる。MOVSBのMOVSBとMOVSWのタイプは、オペランドの指定とタイプの判断を無視して、アセンブラに何を行なうかを知らせる。簡潔で理解しやすいコードのために、前者の方法はしばしば好ましいが、後者は上の例のように効力を無効にする。

```

; MOVE SUBSORT TO TEMP OPERATES BY:
; 1. CALCULATING SUBSORT PTR.
; 2. LOADING POINTER TO RECORD TEMP.
; 3. MOVING BYTES

MOVE$SUBSORT$TO$TEMP  PROC    NEAR
MOV                  AX,SUBSORT$COUNTER
CALL                COMPUTE$POINTER
MOV                  SI,AX
MOV                  DI,OFFSET RECORD$TEMP
REP                  MOVSB          ;COMPUTE POINTER RETURNS
                                           ;CX = 12
RET
MOVE$SUBSORT$TO$TEMP  ENDP

```

前のモジュールと同様に、このコードの実行時間は、

REP      MOVSB

を

SHR      CX,1  
REP      MOVSW

で置き換えることによって減少できる。

## 6.2 I/O ドライバ

I/O ドライバのプログラムのコーディングには次の2つの基本的問題がある。

- これを利用しようとする外部ルーチンによってどのように呼び出されるか。
- このルーチンにパラメータはどのように受け渡されるか。

このルーチンは次の3つの基本的な方法で呼び出される。

- オペレーティング・システムをコールする。これにより、要求は適当なモジュールに受け渡される。
- ドライバ全体のコマンド・ハンドラをコールする。これにより、パラメータが選択されて制御が適当なモジュールに渡される。
- 呼び出し元ルーチンのコード内に存在するアドレスを用いて、直接にルーチンをコールする。

この例では、ルーチンは直接にコールされることを仮定している。呼び出し元プログラムは各ルーチンのエントリ・アドレスを知っている。これは、最初の2つの方法の利用を除外しない。もしこのどちらかが都合がよければ、各モジュールを示す簡単な発送テーブルによって、オペレーティング・システムまたはコマンド・ハンドラはコマンドを適当なモジュールに分配できる。

2章で述べたように、パラメータの受け渡しには3つの方法がある。

- レジスタ
- タスク・ブロック
- スタック

この例では、複数キャラクタの入力と出力のルーチンの場合を除いて、パラメータはレジスタで受け渡される。この機能のために、タスク・ブロックを定義する。複数キャラクタの入力では、次のタスク・ブロックを用いる。

|     |                      |                                                   |
|-----|----------------------|---------------------------------------------------|
| #0: | <input type="text"/> | リードするキャラクタの最大数                                    |
| #1: | <input type="text"/> | 実際にリードしたバイト数                                      |
| #2: | <input type="text"/> | バイト2からnまでは、複数<br>キャラクタの入力ルーチンによって<br>リードされた情報を含む。 |
|     | :                    |                                                   |
| #n: | <input type="text"/> |                                                   |

複数キャラクタの出力では、次のタスク・ブロックを用いる。

#0:  出力すべきバイト数

#1:  バイト1からnは、  
 : データ・ポートのチャン  
 #n:  ネルに送られる

```

CONTROL$PORT      EQU      12H
STATUS$PORT       EQU      12H
DATA$PORT         EQU      10H

; IF BIT 0 OF THE AH REGISTER IS 1, THE USER HAS LOADED SI WITH A
; POINTER TO THE STRING TO BE SENT TO THE CONTROL PORT IF BIT 0 IS A 0, A
; STANDARD INITIALIZATION STRING WILL BE SENT
;

USER$INITIALIZATION$BIT EQU      01H
TIMEOUT$VALUE        EQU      0F000H

; BITS 3, 4, AND 5 OF THE SIO STATUS BYTE ARE ERROR BITS
SIO$ERRORS           EQU      38H
; BIT 1 INDICATES WHETHER OR NOT THE RECEIVER IS READY
; BIT 0 INDICATES WHETHER OR NOT THE TRANSMITTER IS READY
SIO$RECEIVER$READY   EQU      02H
SIO$TRANNNY$READY    EQU      01H
TIMEOUT$ERROR$FLAG   EQU      0FFH

; CARRIAGE RETURN IS TERMINATION CHARACTER FOR READ
CARRIAGE$RETURN      EQU      0DH

; '$' IS TERMINATION CHARACTER FOR WRITE
TERMINATION$CHARACTER EQU      24H
EXTRN SYSTEM$ERROR:  FAR

```

CODE SEGMENT  
 ASSUME CS: CODE

```

; THE INITIALIZATION OPERATES BY:
; 1. TESTING FOR USER SPECIFIED OR
;    SYSTEM INITIALIZATION STRING
; 2. SENDING THE STRING TO THE CONTROL PORT,
;    TERMINATING WHEN A 0 IS DETECTED
;
;
; THIS ROUTINE USES AX AND SI

```

```

INITIALIZATION      PROC      NEAR
                    AND      AH,USER$INITIALIZATION$BIT      ;TEST FOR USER INIT
                    JNZ      SI$LOADED$BY$USER
                    MOV      SI,OFFSET PORT$INITIALIZATION$STRING ;LOAD STANDARD STRING
SI$LOADED$BY$USER:  LODSB
                    OR      AL,AL                              ;SET FLAGS TO TEST FOR 0
                    JZ      DO$A$RETURN                      ;EXIT IF 0
                    OUT      CONTROL$PORT,AL
                    JMP      SI$LOADED$BY$USER
DO$A$RETURN:        RET
INITIALIZATION      ENDP

```

```

PORT$INITIALIZATION$STRING DB      0CEH,40H,0CEH,37H,00H
DO$A$RETURN:            RET
INITIALIZATION          ENDP

```



このルーチンが呼ばれたときの8251は既知の状態にはない事実を考慮すると、4バイトの初期設定ストリングが必要である。たとえば、

CE<sub>16</sub> モード  
37<sub>16</sub> コマンド

の2バイトが8251に送られると、コマンド・コントロール入力を待っている可能性もあるので、8251は正しく初期設定されないときがある。もし

40<sub>16</sub> コマンド (リセット)  
CF<sub>16</sub> モード  
37<sub>16</sub> コマンド

の3バイトが送られると、モード・コントロール入力を待っていた場合には、8251は正しく初期設定されない。しかし、4バイトを用いれば、以前の状態にかかわらず、8251を正しく初期設定できる。

```
; SINGLE CHARACTER INPUT OPERATES BY:
; 1. LOADING TIMEOUT VALUE
; 2. READING THE STATUS PORT AND TESTING FOR SIO ERRORS
; 3. CHECKING FOR TIMEOUT ERRORS
; 4. READING THE DATA
;
; THIS ROUTINE USES AX AND CX
;
; IF ZFLAG IS 1 ON RETURN - ERROR CONDITION
; IF ZFLAG IS 0 ON RETURN - NORMAL OPERATION
; ERROR CONDITIONS RETURNED IN AH
SINGLE$CHARACTER$INPUT      PROC    NEAR
    MOV     CX, TIMEOUT$VALUE
TEST$STATUS:                IN      AL, STATUS$PORT      ;READ STATUS
                           TEST    AL, SIO$ERRORS       ;CHECK FOR ERRORS
                           JNZ     INPUT$ERROR$RETURN
                           DEC     CX                    ;CHECK FOR TIMEOUT
                           JZ      INPUT$TIMEOUT$ERROR$RETURN
                           AND     AL, SIO$RECEIVER$READY ;RECEIVER READY?
                           JZ      TEST$STATUS
                           IN      AL, DATA$PORT        ;GET VALUE
                           RET
INPUT$ERROR$RETURN:         MOV     AH, AL                ;SAVE STATUS
                           XOR     AL, AL                ;SET ZERO FLAG
                           RET
INPUT$TIMEOUT$ERROR$RETURN: MOV     AH, TIMEOUT$ERROR$FLAG ;IF IS TIMEOUT ERROR
                           RET
SINGLE$CHARACTER$INPUT      ENDP
```

```
DEC     CX
JZ      INPUT$TIMEOUT$ERROR$RETURN
AND     AL, SIO$RECEIVER$READY
```

の命令は

```

                                LOOP    NOSTIMEOUT
                                MOV     AH,0FFH                ;TIMEOUT ERROR
                                RET
NOSTIMEOUT:                    AND     AL,SIO$RECEIVER$READY

```

で置き換えられる。これにより、短くて速いオブジェクト・コードが得られる。ただし、ソース・コードの明瞭さを犠牲にしている。

```

; SINGLE CHARACTER OUTPUT OPERATES BY:
; 1. LOADING TIMEOUT VALUE
; 2. READING STATUS PORT
; 3. CHECKING FOR TIMEOUT ERROR
; 4. SENDING DATA TO OUTPUT PORT IF TRANSMITTER IS READY
;
; IF ZFLAG IS 1 ON RETURN - ERROR
; IF ZFLAG IS 0 ON RETURN - NORMAL
;
; THIS ROUTINE USES AX, CX, AND DH
SINGLE$CHARACTER$OUTPUT PROC NEAR
MOV CX,TIMEOUT$VALUE
MOV DH,AL
TRANNY$READY: IN AL,STATUS$PORT
TEST AL,SIO$ERRORS
JNZ OUTPUT$ERROR$RETURN
DEC CX ;TEST FOR TIMEOUT
JZ OUTPUT$TIMEOUT$ERROR$RETURN
AND AL,SIO$TRANNY$READY ;CHECK FOR TRANSMITTER READY
JZ TRANNY$READY

MOV AL,DH GET DATA FROM DH
OUT DATA$PORT,AL
RET
OUTPUT$TIMEOUT$ERROR$ RETURN: MOV AH,TIMEOUT$ERROR$FLAG
RET
OUTPUT$ERROR$RETURN: MOV AH,AL
XOR AL,AL
RET
SINGLE$CHARACTER$OUTPUT ENDP

```

前のモジュールと同様に、タイムアウトのエラー・リターンはメインのコードに含むことができる。

```

DEC CX
JZ OUTPUT$TIMEOUT$ERROR$RETURN
AND AL,SIO$TRANNY$READY

```

を

```

                                LOOP    NOSTIMEOUT
                                MOV     AH,0FFH
                                RET
NOSTIMEOUT:                    AND     AL,SIO$TRANNY$READY

```

で置き換えて、ソース・コードの最後の3行を削除することによって行なえる。

|                                                             |                            |                           |                              |
|-------------------------------------------------------------|----------------------------|---------------------------|------------------------------|
| CHECK\$CHANNEL\$STATUS                                      | PROC<br>IN<br>RET<br>ENDP  | NEAR<br>AL,STATUS\$PORT   | :READ                        |
| CHECK\$CHANNEL\$STATUS                                      |                            |                           |                              |
| SEND\$CONTROL\$INFORMATION                                  | PROC<br>OUT<br>RET<br>ENDP | NEAR<br>CONTROL\$PORT,AL  | :WRITE                       |
| SEND\$CONTROL\$INFORMATION                                  |                            |                           |                              |
| ; MULTIPLE CHARACTER INPUT OPERATES BY:                     |                            |                           |                              |
| ; 1. GETTING # OF BYTES TO READ                             |                            |                           |                              |
| ; 2. CALLING SINGLE CHARACTER INPUT UNTIL                   |                            |                           |                              |
| ;   • ERROR FROM SINGLE CHAR                                |                            |                           |                              |
| ;   • THE MAXIMUM # OF CHARACTERS HAVE BEEN ENTERED         |                            |                           |                              |
| ;   • A TERMINATION CHARACTER (CARRIAGE RETURN) IS ENTERED  |                            |                           |                              |
| ; THIS ROUTINE IS CALLED WITH SI POINTING AT THE TASK BLOCK |                            |                           |                              |
| ;                                                           |                            |                           |                              |
| ;THIS ROUTINE USES SI, DI, AX, CX                           |                            |                           |                              |
| MULTIPLE\$CHARACTER\$INPUT                                  | PROC                       | NEAR                      |                              |
|                                                             | LODSB                      |                           |                              |
|                                                             | OR                         | AL,AL                     | :LOAD MAX # OF BYTES TO READ |
|                                                             | JZ                         | ZERO\$COUNT\$THEN\$RETURN |                              |
|                                                             | MOV                        | DL,AL                     | :SAVE MAX # IN DL            |
|                                                             | MOV                        | DI,SI                     |                              |
|                                                             | INC                        | DI                        | :POINT AT BUFFER             |
| GET\$A\$CHARACTER:                                          | CALL                       | SINGLE\$CHARACTER\$INPUT  | :GET CHARACTER               |
|                                                             | JZ                         | INPUT\$ERROR              |                              |
|                                                             | STOSB                      |                           |                              |
|                                                             | INC                        | BYTE PRT [SI]             | :INCREMENT # READ            |
|                                                             | CMP                        | DL,[SI]                   | :TEST FOR READ MAXIMUM #     |
|                                                             | JZ                         | ZERO\$COUNT\$THEN\$RETURN |                              |
|                                                             | CMP                        | AL,CARRIAGE\$RETURN       |                              |
|                                                             | JNZ                        | GET\$A\$CHARACTER         |                              |
| ZERO\$COUNT\$THEN\$RETURN                                   | RET                        |                           |                              |
| INPUT\$ERROR:                                               | JMP                        | SYSTEM\$ERROR             |                              |
| MULTIPLE\$CHARACTER\$INPUT                                  | ENDP                       |                           |                              |

```

OR      AL,AL
JZ      ZERO$COUNT$THEN$RETURN

```

の命令は、読み込むべきバイト数が0かを調べる。

1つの命令でデータのセーブとポインタの増加を行なうストリング・プリミティブSTOSBのために、DIは入力バッファのポインタとして用いられる。ここでは、ディレクション・フラグは正しく設定されていると仮定している。

|                                                             |       |            |                            |
|-------------------------------------------------------------|-------|------------|----------------------------|
| ; MULTIPLE CHARACTER OUTPUT OPERATES BY:                    |       |            |                            |
| ; 1. GETTING THE NUMBER OF CHARACTERS TO WRITE              |       |            |                            |
| ; 2. CALLING SINGLE CHARACTER OUTPUT UNTIL                  |       |            |                            |
| ;   • ERROR FROM SINGLE CHARACTER OUTPUT                    |       |            |                            |
| ;   • THE MAXIMUM # OF CHARACTERS HAVE BEEN WRITTEN         |       |            |                            |
| ;                                                           |       |            |                            |
| ; THIS ROUTINE IS CALLED WITH SI POINTING AT THE TASK BLOCK |       |            |                            |
| ;                                                           |       |            |                            |
| ; THIS ROUTINE USES AX, SI, DX, AND CX                      |       |            |                            |
| MULTIPLE\$CHARACTER\$OUTPUT                                 | PROC  | NEAR       |                            |
|                                                             | LODSB |            |                            |
|                                                             | OR    | AL,AL      |                            |
|                                                             | JZ    | DO\$RETURN |                            |
|                                                             | MOV   | DL,AL      | :SAVE # OF BYTES TO OUTPUT |

|                             |       |                                            |
|-----------------------------|-------|--------------------------------------------|
| OUTPUT\$\$CHARACTER:        | LODSB |                                            |
|                             | CMP   | AL,TERMINATION\$CHARACTER                  |
|                             | JZ    | DO\$RETURN ;TEST FOR TERMINATION CHARACTER |
|                             | CALL  | SINGLE\$CHARACTER\$OUTPUT                  |
|                             | JZ    | OUTPUT\$ERROR                              |
|                             | DEC   | DL                                         |
|                             | JNZ   | OUTPUT\$\$CHARACTER                        |
| DO\$RETURN:                 | RET   |                                            |
| OUTPUT\$ERROR:              | JMP   | SYSTEM\$ERROR                              |
| MULTIPLE\$CHARACTER\$OUTPUT | ENDP  |                                            |

# 第7章

## 8086 マイクロプロセッサ

この章で初めて、8086と外部ロジックとのインターフェイスの方法について述べる。したがって、この章では、8086が発生または受け取る信号について検討し、8086システムの概念について概要を調べることにする。

この章と以後の章では、5 MHzの標準的な8086のタイミングを主に取り扱っている。A. C. パラメータに適当な代入を行なうことによって、8と10MHzのバージョンにも、示されている式は適用できる。

### 7.1 8086 CPUのピンと信号

8086 CPUのピンと信号を図7-1に示す。特に示されている信号を除いて、すべての入力と出力はTTLレベルと互換性がある。

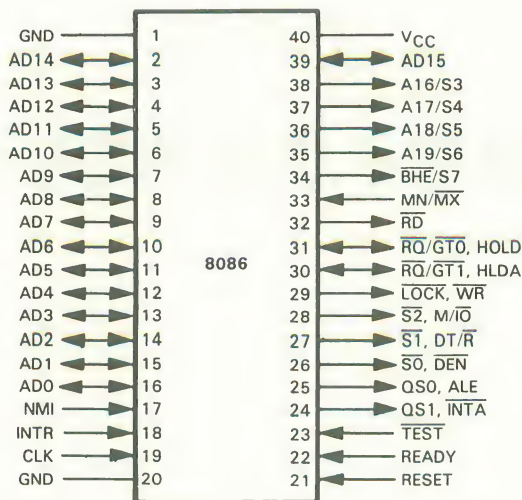
すべてのマイクロプロセッサは、次の種類の信号を生成または受け取る。

- アドレス・ライン
- データ・ライン
- コントロールとステータスのライン
- パワーとタイミングのライン

8086の40ピン・パッケージは、4つのタイプすべての信号を有する。いくつかのピンは、1つ以上のタイプの情報を運ぶ。たとえば、データとアドレスのラインは多重化されている。他のピンは、 $MN/\overline{MX}$ ピンのレベルで定義される機能を持つ。

以下の解説では、 $MN/\overline{MX}$ がハイあるいはローのとき、各ピンの機能について述べている。





| ピンの名前                 | 種 類                         | 型            |
|-----------------------|-----------------------------|--------------|
| AD0-AD15              | データ/アドレス・バス                 | 双方向, トライステート |
| A16/S3, A17/S4        | アドレス/セグメント識別子               | 出力, トライステート  |
| A18/S5                | アドレス/インタラプト・イネーブル・ステータス     | 出力, トライステート  |
| A19/S6                | アドレス/ステータス                  | 出力, トライステート  |
| BHE/S7                | 上位バイト/ステータス                 | 出力, トライステート  |
| RD                    | リード・コントロール                  | 出力, トライステート  |
| READY                 | ウェート状態リクエスト                 | 入力           |
| TEST                  | テストのウェート・コントロール             | 入力           |
| INTR                  | インタラプト・リクエスト                | 入力           |
| NMI                   | ノンマスカブル・インタラプト・リクエスト        | 入力           |
| RESET                 | システム・リセット                   | 入力           |
| CLK                   | システム・クロック                   | 入力           |
| MN/MX                 | マキシマム・システムでは=GND            |              |
| S0, S1, S2            | マシーン・サイクル・ステータス             | 出力, トライステート  |
| RQ/GT0, RQ/GT1        | ローカル・バス・プライオリティ・コントロール      | 双方向          |
| QS0, QS1              | インストラクション・キュー・ステータス         | 出力           |
| LOCK                  | バス・ホールド・コントロール              | 出力, トライステート  |
| MN/MX                 | ミニマム・システムでは=V <sub>CC</sub> |              |
| M/IO                  | メモリまたはI/Oのアクセス              | 出力, トライステート  |
| WR                    | ライト・コントロール                  | 出力, トライステート  |
| ALE                   | アドレス・ラッチ・イネーブル              | 出力           |
| DT/R                  | データのトランスミット/レシーブ            | 出力, トライステート  |
| DEN                   | データ・イネーブル                   | 出力, トライステート  |
| INTA                  | インタラプト・アクノリッジ               | 出力, トライステート  |
| HOLD                  | ホールド・リクエスト                  | 入力           |
| HLDA                  | ホールド・アクノリッジ                 | 出力           |
| V <sub>CC</sub> , GND | パワー, グランド                   |              |

マキシマム・システム信号

ミニマム・システム信号

図7-1 8086のピンと信号の割当て

### 7.1.1 アドレスとデータのライン

8086CPUは、100万（1メガ）バイトのメモリを直接にアドレス指定できる。これは、20ビットのアドレス情報が必要であることを意味する。

8086CPUは、上位バイトと下位バイトとして取り扱う16ビット単位でデータをアクセスする。

20ビットのアドレス・バスと16ビットのデータ・バスを40ピン・パッケージで可能とするために、データ・バスはアドレス・バスの下位16ビットと多重化されている。さらに4つのアドレス・ラインはステータス情報と多重化されている。アドレス/データとアドレス/ステータスの信号は次のとおりである。

AD0-AD15  
A16/S3  
A17/S4  
A18/S5  
A19/S6  
BHE/S7

これらのラインについて詳細に検討してみる。

**AD0-AD15.** この16本のラインは、アドレス・バスとデータ・バスの多重化されたラインである。バス・サイクルの最初のクロックの期間、このラインはアドレスの下位16ビットを含む。他のすべてのクロック・サイクルの間、このラインはデータ・バスとして用いられる。このラインは、8086がインタラプト・アクノリッジのサイクルあるいは“ホールド・アクノリッジ”のサイクルを実行しているときは、ハイ・インピーダンスの状態に置かれる。

**A16/S3.** 命令実行の最初のクロックの期間、このラインはアドレス16のラインとして動作する。I/O命令が実行されると、このラインは最初のクロックの期間、ローになる。他のすべてのクロックの期間、このラインはA17/S4と共にステータス情報を得るために用いられる。

**A17/S4.** 命令実行の最初のクロックの期間、このラインはアドレス17のラインとして動作する。I/O命令が実行されると、このラインは最初のクロックの期間、ローになる。他のサイクルでは、このラインはA16/S3と共にステータス情報を得るために用いられる。

最初を除くすべてのクロックの期間、A16/S3とA17/S4は、次に示すように、どのセグメント・レジスタが8086のアドレスのセグメント部を生成しているかを示す情報を与える。

| A 17/S4 | A 16/S3 | 意 味                  |
|---------|---------|----------------------|
| 0       | 0       | エキストラ・セグメント          |
| 0       | 1       | スタック・セグメント           |
| 1       | 0       | コード・セグメントまたはセグメントでない |
| 1       | 1       | データ・セグメント            |

この情報は、各セグメント・レジスタがそれぞれ独自の1メガバイトのメモリを指定するように、外部ロジックによって8086のメモリ領域を拡張するために利用できるが、このとき、異なるセグメントで計算されるメモリ・アドレスはオーバーラップすることができなくなる。

**A18 / S5** . 命令実行の最初のクロックの期間、このラインはアドレス18のラインとして動作する。I/O 命令が実行されると、このラインは最初のクロックの期間、ローになる。他のすべてのクロックの期間、このラインは8086のインタラプト・イネーブル・フラグの状態を示している。

**A19 / S6** . 命令実行の最初のクロックの期間、このラインはアドレス19のラインとして動作する。I/O 命令が実行されると、このラインは最初のクロックの期間、ローになる。他のすべてのサイクルでは、8086は、システム・バスを制御していれば、このラインをローに保つ。“ホールド・アクノリッジ”の期間、他のバス・マスタがシステム・バスの制御を得られるように、8086はこのラインをフロート状態にする。

**$\overline{\text{BHE}}$  / S7** . 命令実行の最初のクロックの期間、このラインは $\overline{\text{BHE}}$ として用いられる。データ・バスの上位8ビットでデータが転送される、読み出し、書き込み、そしてインタラプト・アクノリッジのシーケンスの間、 $\overline{\text{BHE}}$ はローに保たれる。この信号は、AD0のラインと共にメモリ・バンクを選択するために用いられる。8086のメモリ選択のより一般的な解説は次の節に示す。2番目以降のクロックの期間では、 $\overline{\text{BHE}}$  / S7は最初のクロックにおける出力レベルを維持する。

### 7.1.2 コントロールとステータスのライン

8086のコントロールとステータスのラインは2つに分類することができる。1つはMN /  $\overline{\text{MX}}$ ピンのレベルの影響を受けないものであり、もう1つはその機能がMN /  $\overline{\text{MX}}$ ピンの値に依存するものである。影響を受けないものには次のものがある。

$\overline{\text{RD}}$   
READY  
 $\overline{\text{TEST}}$   
INTR  
NMI  
RESET

$\overline{\text{RD}}$ は、CPUがメモリあるいはI/O素子からデータを読み出すとき、ローが出力される。 $\overline{\text{S2}}-(\text{M}/\text{I/O})$ ピンは、メモリまたはI/Oのどちらが要求されているかを指定する。

READYは、データ転送操作を行なう用意のできていることを示すために、選ばれたメモリまたはI/O素子によって用いられる。信号( $\overline{\text{RDY1}}$ または $\overline{\text{RDY2}}$ )は8284クロック・ジェネレータに入力され、READYの入力はクロックとの同期がとられる。適当なときにREADYにローが入力されると、READYがハイになるまで、8086は“ウェート”状態となる。 $\overline{\text{TEST}}$ は、8086のWAIT命令だけで用いられる入力である。WAIT命令が実行されると、8086は $\overline{\text{TEST}}$ にローが入力されるまで休止する。

**INTR**は、割り込み要求の入力である。この信号は、各々の命令実行の最終クロックの期間に8086によってサンプルされる。インタラプト・イネーブル・ビットが1で**INTR**がハイならば、8086はインタラプト・アクノリッジ・シーケンスを実行して、制御を適当なインタラプト・サービス・ルーチンに移す。それ以外では、次の命令が実行される。**INTR**はレベル・トリガの入力である。

**NMI**は、ノンマスカブル・インタラプト要求の入力である。**NMI**はエッジ・トリガの入力である。**NMI**がローからハイになれば、8086は現在の命令の実行を終了して、制御をノンマスカブル・インタラプト・サービス・ルーチンに移す。ノンマスカブル・インタラプト・サービス・ルーチンのアドレスはメモリ位置 00008<sub>16</sub>に存在する。ソフトウェアではこのインタラプトを無効にできない。

**RESET**はシステム・リセット信号であり、パワーアップ時を除いて**RESET**が少なくとも50 $\mu$ s 持続しなければならないときは、少なくとも4つのCLKクロックの期間、8284クロック・ジェネレータにハイが入力される必要がある。8284は**RESET**の同期をとり、8086に転送する。**RESET**がローに戻ると、次のことが起きる。

1. フラグ・レジスタが0000<sub>16</sub>に設定される。これはインタラプトとシングル・ステップ・モードを無効にする作用をもつ。
2. DS, SS, ES, さらにPCのレジスタは0000<sub>16</sub>にリセットされる。
3. CSレジスタはFFFF<sub>16</sub>に設定される。

実行はメモリ位置 FFFF0<sub>16</sub> から続行される。

MN/ $\overline{\text{MX}}$ の影響を受ける信号には次のものがある。

マックス ミニ  
 $\overline{\text{S0}}-(\overline{\text{DEN}})$   
 $\overline{\text{S1}}-(\text{DT}/\overline{\text{R}})$   
 $\overline{\text{S2}}-(\text{M}/\overline{\text{IO}})$   
 $\overline{\text{RQ}}/\overline{\text{GT0}}-(\text{HOLD})$   
 $\overline{\text{RQ}}/\overline{\text{GT1}}-(\text{HLDA})$   
 $\text{QS0}-(\text{ALE})$   
 $\text{QS1}-(\overline{\text{INTA}})$   
 $\overline{\text{LOCK}}-(\overline{\text{WR}})$

MN/ $\overline{\text{MX}}$ が接地されていると、8086は“マキシマム・モード”にあると呼ばれる。MN/ $\overline{\text{MX}}$ がハイのとき、8086は“ミニマム・モード”にあると呼ばれる。

$\overline{\text{S0}}-(\overline{\text{DEN}})$ 。MN/ $\overline{\text{MX}}$ ピンが接地されていると、このピンは $\overline{\text{S0}}$ として機能する。 $\overline{\text{S0}}$ は、 $\overline{\text{S1}}-(\text{DT}/\overline{\text{R}})$ と $\overline{\text{S2}}-(\text{M}/\overline{\text{IO}})$ と共にステータス情報を得るために用いられる。このステータス情報については、 $\overline{\text{S2}}-(\text{M}/\overline{\text{IO}})$ の記述に続いて論じる。MN/ $\overline{\text{MX}}$ ピンのレベルがハイならば、このピンは $\overline{\text{DEN}}$ として機能する。 $\overline{\text{DEN}}$ は、(DT/ $\overline{\text{R}}$ で決められる)システムあるいはローカルのバスへのバッファのデータ・トランシーバをイネーブルにすることによって、8286/8287バッファを制御するために用いられる。

$\overline{\text{S1}}-(\text{DT}/\overline{\text{R}})$ 。MN/ $\overline{\text{MX}}$ ピンが接地されていると、このピンは $\overline{\text{S1}}$ として機能する。 $\overline{\text{S1}}$ は、 $\overline{\text{S0}}-(\overline{\text{DEN}})$ と $\overline{\text{S2}}-(\text{M}/\overline{\text{IO}})$ と共にステータス情報を得るために用いられる。



この情報については、 $\overline{S2}-(M/\overline{IO})$  の記述に続いて論じる。MN/ $\overline{MX}$ ピンのレベルがハイならば、このピンはDT/ $\overline{R}$ として機能する。DT/ $\overline{R}$ は、データ伝送方向を示して、8286/8287バッファをコントロールするために用いられる。DT/ $\overline{R}$ がハイならば、トランシーバはシステム・バスにデータを設定し、DT/ $\overline{R}$ がローならば、トランシーバはシステム・バスからデータを取り去る。8288バス・コントローラはまた $\overline{DEN}$ とDT/ $\overline{R}$ の出力を生成する。バス・コントローラが存在すれば、その $\overline{DEN}$ とDT/ $\overline{R}$ の出力は、8086の $\overline{DEN}$ とDT/ $\overline{R}$ の出力の代わりに用いられる。この異なる構成については後で述べる。

$\overline{S2}-(M/\overline{IO})$ 。MN/ $\overline{MX}$ ピンが接地されていると、このピンは $\overline{S2}$ として機能する。 $\overline{S2}$ は、 $\overline{S0}-(\overline{DEN})$ と $\overline{S1}-(DT/\overline{R})$ と共に以下に述べるステータス情報を得るために用いられる。MN/ $\overline{MX}$ ピンのレベルがハイならば、このピンは $M/\overline{IO}$ として機能する。メモリまたはI/Oのアクセスの間、 $M/\overline{IO}$ はメモリ・アクセスに対してはハイに、I/Oアクセスに対してはローになる。

MN/ $\overline{MX}$ が接地されていると、次のように $\overline{S0}$ 、 $\overline{S1}$ 、さらに $\overline{S2}$ によって、8288バス・コントローラにステータスが供給される。

| $\overline{S2}$ | $\overline{S1}$ | $\overline{S0}$ |                |
|-----------------|-----------------|-----------------|----------------|
| 0               | 0               | 0               | インタラプト・アクノリッジ  |
| 0               | 0               | 1               | I/O リード        |
| 0               | 1               | 0               | I/O ライト        |
| 0               | 1               | 1               | ホルト            |
| 1               | 0               | 0               | インストラクション・フェッチ |
| 1               | 0               | 1               | メモリ・リード        |
| 1               | 1               | 0               | メモリ・ライト        |
| 1               | 1               | 1               | インアクティブ        |

この情報は、マキシム・モード・システムのためのメモリとI/Oのコントロール信号を生成するために、8288バス・コントローラによって用いられる。

$QS0-(ALE)$ 。MN/ $\overline{MX}$ ピンが接地されていると、このピンは $QS0$ として機能する。 $QS0$ は、 $QS1-(\overline{INTA})$ と共に8086の命令キューの状態を得るために用いられる。詳細は後で述べるが、命令キューは8086マイクロプロセッサで6バイトの領域がある。これは、実行を待つオブジェクト・コード・バイトの保持に用いられる。MN/ $\overline{MX}$ ピンのレベルがハイであれば、 $QS0-(ALE)$ は $ALE$ として機能する。有効なメモリ・アドレスがアドレス/データ・バスに存在する間は、ハイの $ALE$ パルスが出力される。マキシマム・モード・システムで、 $ALE$ は8288バス・コントローラによって与えられる。

$QS1-(\overline{INTA})$ 。MN/ $\overline{MX}$ ピンが接地されていると、このピンは $QS1$ として機能する。 $QS1$ は、 $QS0-(ALE)$ と共に、以下に述べるように8086の命令キューの状態を得るために用いられる。MN/ $\overline{MX}$ ピンのレベルがハイならば、このピンは $\overline{INTA}$ として機能する。

8086がインタラプト・アクノリッジ・シーケンスを実行している間、 $\overline{INTA}$ はローになる。マキシマム・システムで、 $\overline{INTA}$ は8288バス・コントローラによって与えられる。MN/ $\overline{MX}$ ピンが接地されていると、8086の命令キューの状態は、次のように $QS0$ と $QS1$ によって与えられる。



| QSO | QS1 |                         |
|-----|-----|-------------------------|
| 0   | 0   | ノー・オペレーション              |
| 0   | 1   | 命令の最初のバイトが実行されている。      |
| 1   | 0   | キューが空になっている。            |
| 1   | 1   | 命令の次のバイトがキューから取り出されている。 |

QSOとQS1は、キュー操作に続くクロックの期間、有効となる。

$\overline{RQ/GT0}$ -(HOLD).  $MN/\overline{MX}$ ピンが接地されていると、このピンは $\overline{RQ/GT0}$ として機能する。 $\overline{RQ/GT0}$ は要求/許可のラインである。他のバス・マスタは、このピンにローのパルスを入力することによって、8086をHOLD状態にできる。8086は、HOLD状態になったことを、 $\overline{RQ/GT0}$ でローのパルスを出力して、要求元のバス・マスタに通知する。次いで8086は、システム・バスとスリーステート出力の制御を放棄する。新しいバス・マスタが次にシステム・バスの制御を放棄すると、もう1つのローの $\overline{RQ/GT0}$ パルスを送って同様のことを行なう。そして8086はバス制御を再び確立する。要求/許可のシーケンスは8章で詳細に述べる。 $MN/\overline{MX}$ ピンのレベルがハイならば、 $\overline{RQ/GT0}$ -(HOLD)はHOLDとして機能する。HOLDは、外部ロジックによってHOLDのリクエスト・ラインとして用いられる。外部ロジックがHOLDのレベルをハイにすると、8086は現在のバス・サイクルを完了してHOLD状態になる。8086は、HLDAにハイを出力してHOLD状態となったことを通知する。

$\overline{RQ/GT1}$ -(HLDA).  $MN/\overline{MX}$ ピンが接地されていると、このピンは $\overline{RQ/GT1}$ として機能する。 $\overline{RQ/GT1}$ は、 $\overline{RQ/GT0}$ よりもプライオリティが低いことを除いて、機能的には $\overline{RQ/GT0}$ と同一である。 $\overline{RQ/GT0}$ の要求/許可のシーケンスが進行中でなければ、8086は $\overline{RQ/GT1}$ の要求/許可のシーケンスを開始することができる。要求/許可のシーケンスの詳細は8章に述べられている。 $MN/\overline{MX}$ ピンのレベルがハイならば、 $\overline{RQ/GT1}$ -(HLDA)はHLDAとして機能する。HLDAはHOLDアクノリッジの信号である。HLDAは、HOLDによって作られたホールド要求を確認するために、ハイが出力される。HLDAの信号がハイになると、8086CPUはまた、そのスリーステートの出力信号をフロート状態にする。したがって、システム・バスをフロート状態とする。

$\overline{LOCK}$ -( $\overline{WR}$ ).  $MN/\overline{MX}$ ピンが接地されていると、このピンは $\overline{LOCK}$ として機能する。 $\overline{LOCK}$ は、命令実行中に8086がシステム・バスの制御を失なうことを避けるために、ローが出力される。 $\overline{LOCK}$ がローの間、外部ハードウェアは、他のバス・マスタがシステム・バスの制御を獲得しないことを保証しなければならない。8086が $\overline{LOCK}$ 命令を実行すると、次の命令が実行される間、 $\overline{LOCK}$ はローが出力される。 $MN/\overline{MX}$ ピンのレベルがハイならば、 $\overline{LOCK}$ -( $\overline{WR}$ )は $\overline{WR}$ として機能する。 $\overline{WR}$ は、メモリまたはI/Oのライトの間、ローが出力される。アドレス/データ・バスに出力されているデータが安定なときに、パルスの立下りエッジが生じる。

### 7.1.3 パワーとタイミングのライン

CLKは、すべての8086ロジックを同期させるために用いられるクロック信号である。

この信号は普通、8284クロック・ジェネレータによって出力される。

Vcc はパワー・サプライのピンである。8086は、このピンに $+5\text{ V} \pm 10\%$ が加えられる必要がある。

2つのGNDピンが存在し、これらは共に接地のピンである。

#### 7.1.4 スリーステートのラインと信号

次の8086の信号はスリーステートである。

AD0-AD15  
A16/S3  
A17/S4  
A18/S5  
A19/S6  
BHE/S7  
RD  
S0-(DEN)  
S1-(DT/R)  
S2-(M/I0)  
LOCK-(WR)  
INTA

8086がHOLD状態にある間、これらすべての信号はハイ・インピーダンス状態になる。 $\overline{S0}-(\overline{DEN})$ 、 $\overline{S1}-(\overline{DT/R})$ 、 $\overline{S2}-(\overline{M/I0})$ の信号は、8086がホールド・アクノリッジを出す直前に、フロート状態になる。

インタラプト・アクノリッジの間、AD0-AD15、A16/S3、A17/S4、A18/S5、A19/S6のラインはフロート状態になる。

## 7.2 8086の概要と基本的システムの概念

この節では、バス・サイクル、アドレス/データ・バス、システム・データ・バス、エグゼキューション・ユニット、バス・インターフェイス・ユニット、命令キューを含む基本的な8086のシステムの概念について論じる。

### 7.2.1 8086バス・サイクルの定義

8086は、システム・バスを通して外部ロジックと連絡する。8086は、データを伝送または命令をフェッチするために、“バス・サイクル”を実行する。“バス・サイクル”を図7-2に示す。

最小のバス・サイクルは、Tステートと呼ばれる4つのCPUクロックの期間より成る。最初のTステート(T1)の間、8086は、20ビットの多重化アドレス/データ/ステータス・バスにアドレスを出力する。バス上のアドレスは、ALE信号がハイからローに移移するとき、有効とみなされる。ミニマム・システムでは、この信号は8086によって作られる。マキシマム・システムでは、この信号は8288バス・コントローラによって作られる。

$\overline{S2}-(M/\overline{IO})$  信号は、メモリまたはI/Oのアクセスのどちらが実行されているかを示す。

第2のTステート(T2)の間、8086はアドレス・バスからアドレスを取り去る。リード・バス・サイクルに対しては、リード・サイクルの準備としてデータ・バス・ラインはフロート状態になる。ライト・バス・サイクルに対しては、データ・バス・ラインにデータが出力される。

データ・バス・トランシーバは、8086のシステム構成と伝送の方向(8086へか8086からか)に依存して、T1またはT2の間、イネーブルとなる。読み出し、書き込み、またはインタラプト・アクノリッジのコントロール信号はT2の間、常に有効である。

T2の間、バス・サイクル・ステータス(S3, S4, S5, S6)は、上位4つのアドレス/ステータス・バス・ラインに出力される。次に示すステータス情報は、残りのバス・サイクルで有効である。

| S4                         | S3 |                                      |
|----------------------------|----|--------------------------------------|
| 0                          | 0  | エキストラ(ESセグメントに対する相対)                 |
| 0                          | 1  | スタック(SSセグメントに対する相対)                  |
| 1                          | 0  | コード/なし(CSセグメントに対する相対<br>またはデフォルト・ゼロ) |
| 1                          | 1  | データ(DSセグメントに対する相対)                   |
| S5=IF (インタラプト・イネーブル・フラグ)   |    |                                      |
| S6=0 (8086がバスを使用していることを示す) |    |                                      |

8086は、T3の間、上位4つのアドレス/ステータス・バス・ラインに、ステータス情報を出し続ける。また、8086はライト・バス・サイクルの間、データを出力し続ける。リード・バス・サイクルに対して、8086はT3の終わりにデータの入力を行なう。選択された素子に必要な速度でデータの伝送を行なう能力がなければ、素子はREADYにローを入力して“ノット・レディ”を知らせる必要がある。

これにより、8086はT3の後に付加的なクロック・サイクルを挿入する。この付加的なクロック・サイクルは、Twステート(ウェート・ステート)と呼ばれる。“ノット・レディ”の表示は、T3の開始前にCPUに示されなければならない。Twの間のバス動作はT3と同じである。選ばれた素子が伝送完了の十分な時間を有していれば、READYをハイにする。Twクロック期間が終わった後で、T4、すなわちバス・サイクルの最後のクロック期間が実行される。

T4の間に、メモリとI/Oのコントロール・ラインはディスエーブルとなり、選ばれた外部素子はシステム・バスから切り離される。

図7.2 に基本的な8086バス・サイクルを示す。

バス・サイクルは、システム・バス上の素子に対して、素子と素子内のレジスタまたはメモリ位置を選択するためのアドレス、さらにデータに伴うリード・ストローブまたはライト・ストローブより成る非同期の事象に見える。選択された素子は、ライト・サイクルの間にデータを受け取り、リード・サイクルの間にバス上にデータを設定しなければなら

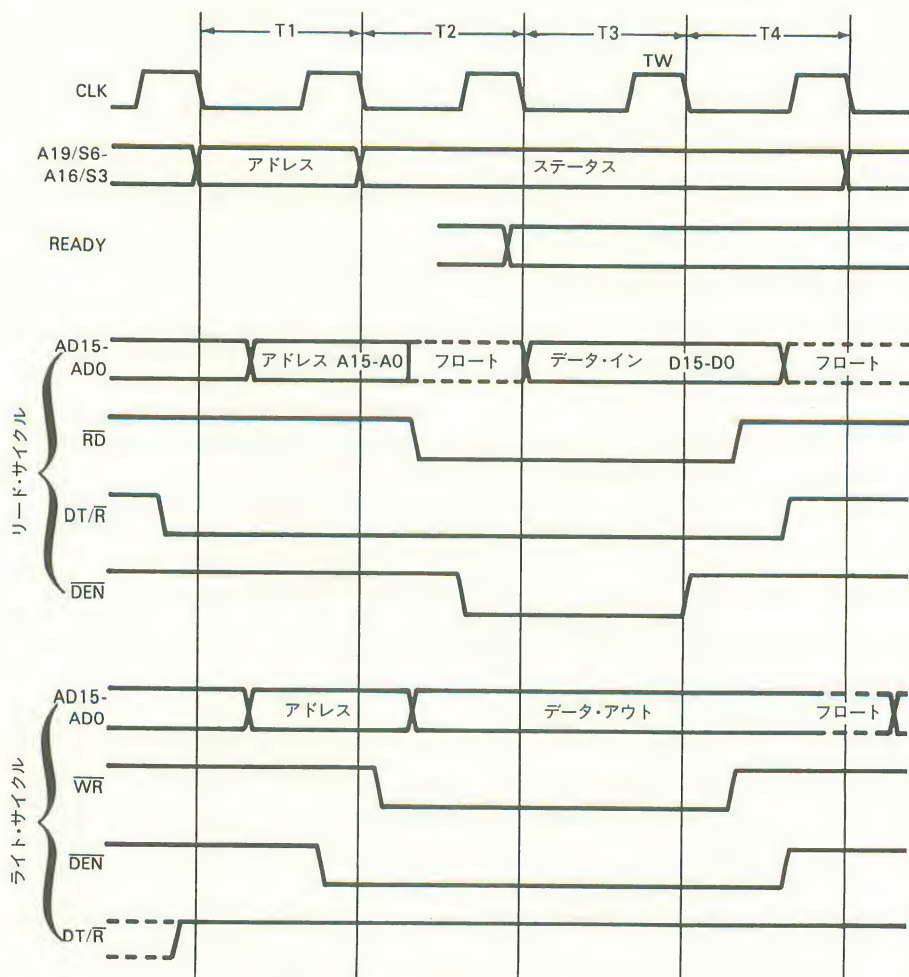


図7-2 基本的な8086のバス・サイクル

ない。バス・サイクルの終わりに、素子を書かれるデータをラッチまたはバス・ドライバをデイスエーブルにする。素子がバス・サイクルを変更できるただ1つの方法は、READYのコントロール・ラインによってウエート状態のクロック期間を挿入することで行なえる。

命令のフェッチが行なわれているか、あるいは8086とメモリまたはI/O素子の間でオペランドが伝送されるべきときは、8086は単にバス・サイクルを実行するだけである。バス・サイクルが実行されていないときは、8086のバス・インターフェイス・ロジックはアイドル・クロック期間 (T<sub>I</sub> と呼ぶ) を実行する。アイドル・クロックの期間、8086は以前のバスサイクルから上位4つのアドレス・ラインにステータス情報を出力し続ける。以前のバス・サイクルが書き込みならば、次のバス・サイクルの開始までCPUは16個のデータ・バス・ラインにデータを出力し続ける。8086がリード・サイクルに続いてアイドル・ク



ロック期間を実行するならば、次のバス・サイクルの開始まで8086は16個のデータ・バス・ラインをフロート状態にする。

メモリのアクセスに際して、8086は2つのタイプの操作を行なう。

- 命令のフェッチ

●命令に必要なオペランドの読み込みまたは書き込みを行なうためのメモリ・アクセス  
命令フェッチとメモリ・アクセスのバス・サイクルの間の、通常簡単な連続した関係は、8086内では6バイトの命令オブジェクト・コードのキューの存在によって変えられている。

8086のバス・インターフェイス・ロジックがアイドルでなければ、命令フェッチのバス・サイクルを実行する代わりに、命令キューが満たされるまで、プログラム・メモリから次に続くオブジェクト・コード・バイトをフェッチする。1つ以上の命令のオブジェクト・コードがキューにあれば、命令フェッチのバス・サイクルは、付加的な命令フェッチのバス・サイクルによってオペランドのメモリ・アクセス・バス・サイクルと分離される。

ジャンプまたはコールの命令が実行されると、現在の命令キューにある次に続く命令のオブジェクト・コード・バイトはもはや必要でなくなる。したがって、キューの内容は何の悪影響もなく捨てられる。

## 7.2.2 8086のアドレスとデータ・バスの概念

8086に接続されるほとんどのメモリ素子と周辺装置は、全バス・サイクルの間、安定なアドレスを必要とするので、T1の間に多重化されたアドレス/データ・バス上に存在するアドレスはラッチされなければならない。このラッチされたアドレスは、必要な周辺装置またはメモリ位置を選択するために用いられる。アドレス/データ・バスの分離のために、8086にはアドレス・ラッチ・イネーブル信号(ALE)があり、これは8282あるいは8283の8ビット双安定ラッチにアドレスを捕えるために用いることができる。

このラッチはインパート(8283)またはノンインパート(8282)であり、32mAのドライブ能力を持ち、22ns(インパート)または30ns(ノンインパート)で300pFの容量負荷をスイッチできるトライステート・バッファによって駆動される出力を有する。8282/8283ラッチは、ALEがハイの間、アドレスを出力に伝え、ALEの次のエッジでアドレスをラッチする。しかしこれは、ラッチの伝播遅延で、アドレス・アクセスとチップ・セレクトのデコードを遅らせる。

図7.3 にアドレス/データ・バスの分離を示す。

ラッチの出力は、ロー・アクティブの $\overline{OE}$ 入力によってイネーブルとなる。多重化アドレス/データ・バスの分離(多重化バスからのアドレスのラッチ)は、単独のアドレス・バスがシステム全体にアドレスを分配して、システムの適当な時点またはCPUで局所的に処理できる。

最適なシステムの性能と、マルチプロセッサとMultibus構成との互換性から、図7-4に示す局所的信号分離は、図7-5に示されている分散信号分離よりも強く推薦される。この章の残りでは、図7-4で示しているように、バスはCPU側で分離されることを仮定している。

8086のメモリ・アドレス領域は、バイトが8ビットのデータ要素を含み、2つの連続す



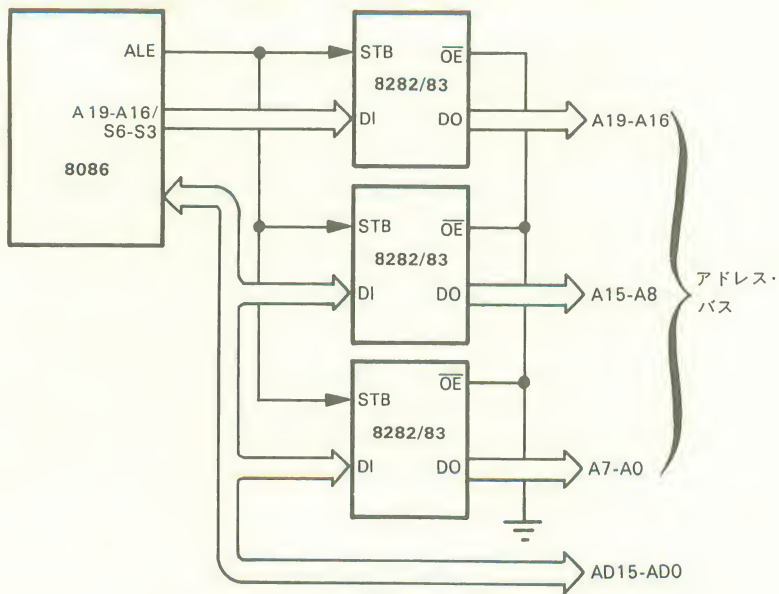


図7-3 アドレス / データ・バスの分離

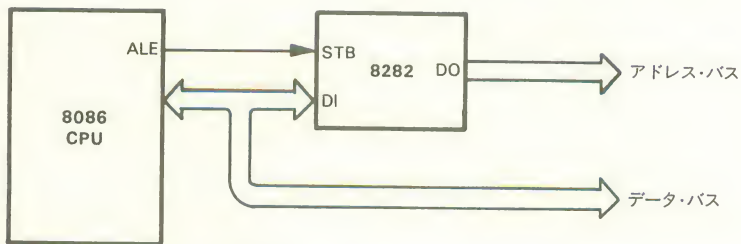


図7-4 別々のアドレスとデータのバス

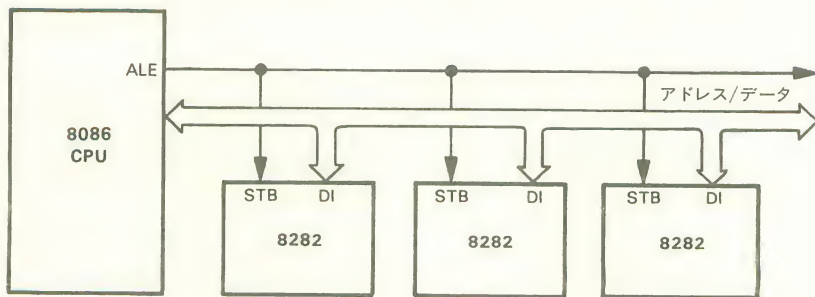
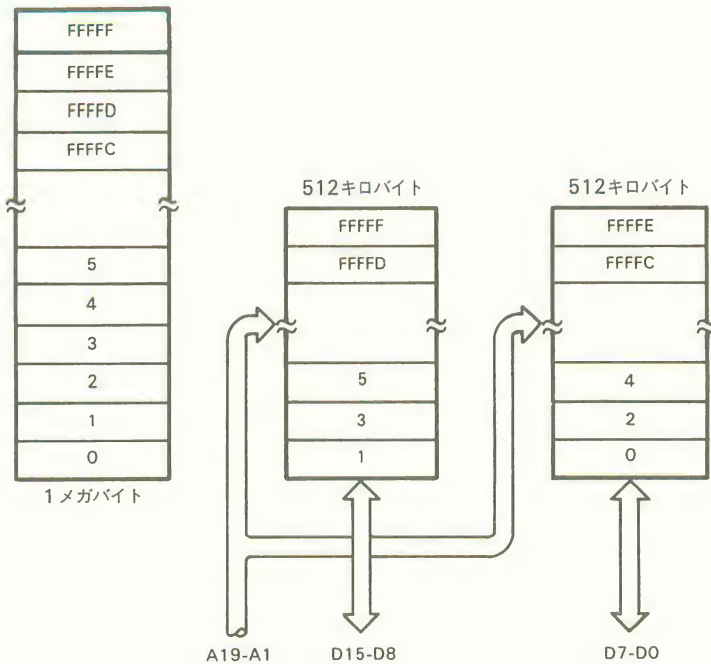
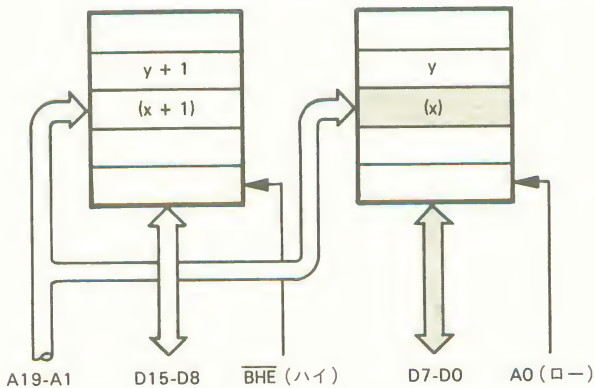


図7-5 局所的アドレスの分離の多重化バス



るバイトが16ビットのデータ要素を含む連続した1メガバイトとして考えられる。バイトまたはワードのアドレス境界に何の制限も存在しない。アドレス領域は、この領域を各々512キロバイトまでの2つの8ビットのバンクに分けることによって、16ビットのデータ・バスに物理的に接続されている。

1つのバンクは、16ビット・データ・バスの下位(D7-0)に接続されていて、偶数アドレス・バイト( $A_0 = 0$ )を含む。もう1つのバンクは、データ・バスの上位(D15-8)に

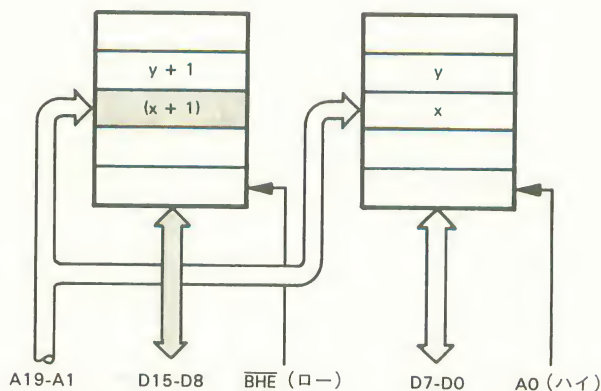


接続されていて、奇数アドレス・バイトを含む( $A0=1$ )。各バンクの特定のバイトは、アドレス・ライン  $A19-A1$  によって選択される。データ・バスの下位 ( $D7-0$ ) を通して偶数アドレスにデータ・バイトは伝送される。

$A0$ はローのとき、データ・バスの下位に接続されているメモリ・バンクを選択し、バス・ハイ・イネーブル( $\overline{BHE}$ )はハイを出力してデータ・バスの上位のメモリ・バンクをディスエーブルにする。このディスエーブル操作は、下位メモリ・バンクへの書き込み操作で上位バンクのデータを破壊することを防ぐために必要である。 $\overline{BHE}$ もまた、 $A19-A16$ アドレス・ラインと同じタイミングで多重化された信号なので、全バス・サイクルの間、安定な信号を得るために  $ALE$  によってラッチされなければならない。

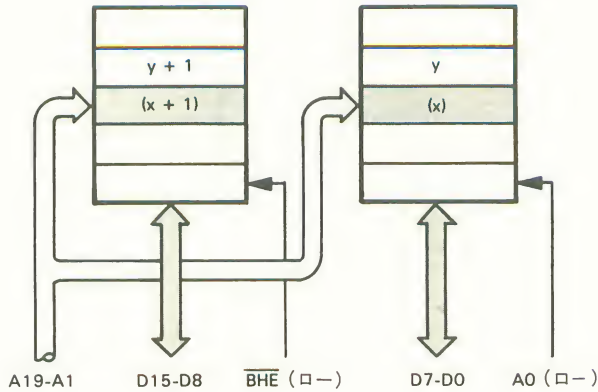
$T2$  から  $T4$  の間、 $\overline{BHE}$ の出力はステータス・ライン  $S7$  で利用できる (ステータス・ライン  $S7$  の意味はまだ定義されていない)。

奇数アドレスのメモリ・バイトにアクセスすると、情報はデータ・バスの上位 ( $D15-D8$ ) を通して伝送される。 $\overline{BHE}$ は上位メモリ・バンクを有効にするためにローが出力される。 $A0$ は下位メモリ・バンクをディスエーブルにするためにハイが出力される。これは次のように図示される。



8086は、データ・バスの適正な片方を通してデータを伝送し、必要な信号レベルの  $\overline{BHE}$  と  $A0$  を出力する。

例として、奇数アドレスのメモリ位置から  $CL$  レジスタ ( $CX$  レジスタの下位) に1バイトのデータをロードすることを考える。このデータは、16ビット・データ・バスの上位を通してアクセスされる。このデータは上位8つのデータ・バス・ラインによって8086に伝送されるが、8086は自動的にデータの方角を内部の16ビット・データ・バスの下位、したがって  $CL$  レジスタに変える。この能力により、 $AL$  レジスタによるバイトのI/O伝送の、16ビット・データ・バスの上位あるいは下位に接続されているI/O素子へのアクセスが可能となる。偶数アドレスに位置する16ビットのワード(下位バイトが偶数バイト・アドレスの連続した2バイト)は、1つのバス・サイクルでアクセスされる。 $A19-A1$ は各バンクの適当なバイトを選択し、 $A0$ のローと  $\overline{BHE}$  のローによって両方のバンクが同時にイネーブ



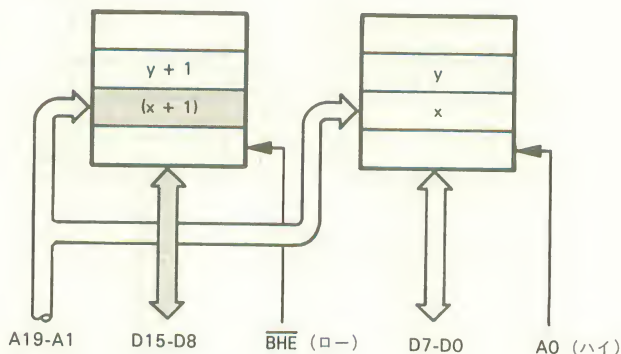
ルとなる。

奇数アドレスに位置する16ビットのワード（下位バイトが奇数バイト・アドレスの連続した2バイト）は、2つのバス・サイクルでアクセスされる。最初のバス・サイクルで、下位バイト（奇数バイト・アドレス）がアクセスされる。第2のバス・サイクルで、上位バイト（偶数バイト・アドレス）がアクセスされる。最初のバス・サイクルの間、 $A_{19}-A_1$ はアドレスを指定する。 $A_0$ は1（奇数アドレス）で $\overline{BHE}$ はローである。したがって、下位のメモリ・バンクはディスエーブルとなり、上位メモリ・バンクはイネーブルとなる。第2のバス・サイクルでは、アドレスは増加する。したがって $A_0$ は0となる。しかし $\overline{BHE}$ はハイなので、下位のメモリ・バンクはイネーブルとなり、上位のメモリ・バンクはディスエーブルとなる。これは次ページに示した図のようになる。

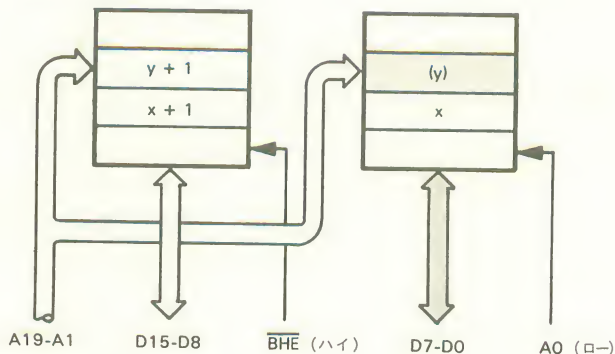
上に示した一連の過程は、奇数アドレスのワード伝送が指定されたときは常に、8086によって自動的に実行される。8086は、内部の16ビット・レジスタの上位と下位のバイトと、データ・バスの適当な部分とを自動的に接続する。ただし、奇数アドレス境界のワードにアクセスするには余分のバス・サイクルを必要とすることに注意。これはシステムの性能を低下させる。

バイト・リードの期間、データ・バスの上位または下位で両方ではない位置にデータが予期されても、CPUはクロック期間T2の間、16ビット・データ・バスのすべてをフロート状態にする。これにより、読み出し専用素子（ROM、EPROM）に対するチップ・セレクトのデコードの条件が簡単になる（チップ・セレクトのロジックについては後で述べる）。バイト・ライト操作の間、8086は16ビット・データ・バスのすべてを駆動する。データが伝送されていない方のデータ・バスの情報は不定である。このような概念はI/Oアドレス領域にも適用される。

最初のパス・サイクル



第2のパス・サイクル



### 7.2.3 システム・データ・バスの概念

システム・データ・バスを取り上げると、(a)図7-6に示す多重化アドレス/データ・バス、あるいは(b)図7-7 (374 ページ) に示すような、トランシーバによって多重化バスにバッファを用いたデータ・バスの2つの実現のどちらかを考えなければならない。

多重化データ・バスを用いるとき、設計者は多重化バスに直接接続されるメモリまたはI/O素子がT<sub>1</sub>の期間にバス上のアドレスを壊さないことを保証しなければならない。この状況を避けるために、図7-8に示されているように、素子の出力ドライバは、素子チップ・セレクトによってイネーブルとなつてはならず、システムのリード信号によって出力のイネーブルが制御されるものでなければならない。



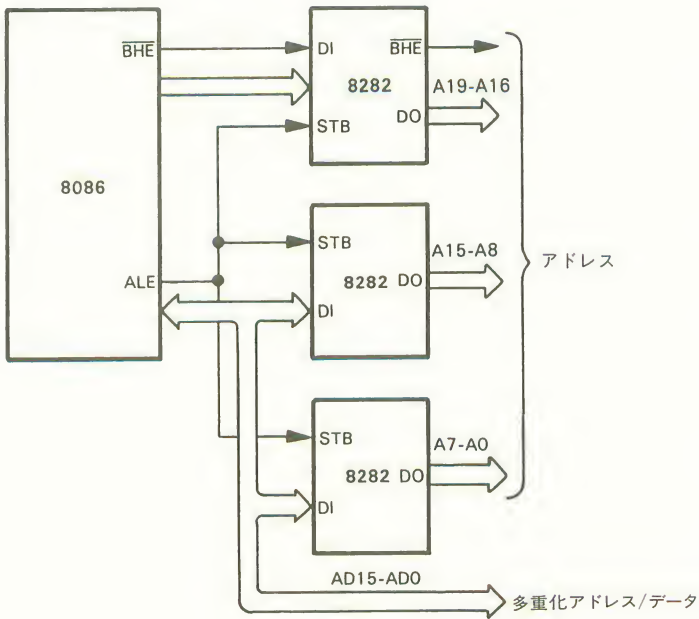


図7-6 多重化データ・バス

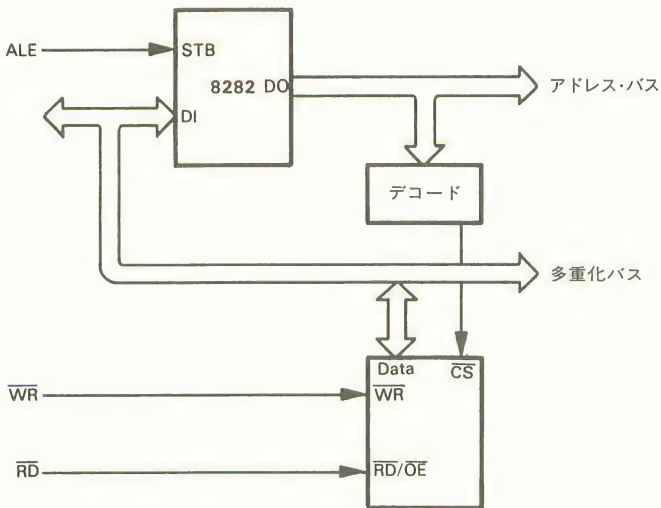


図7-8 多重化バス上のアウトプット・イネーブルを持つ素子

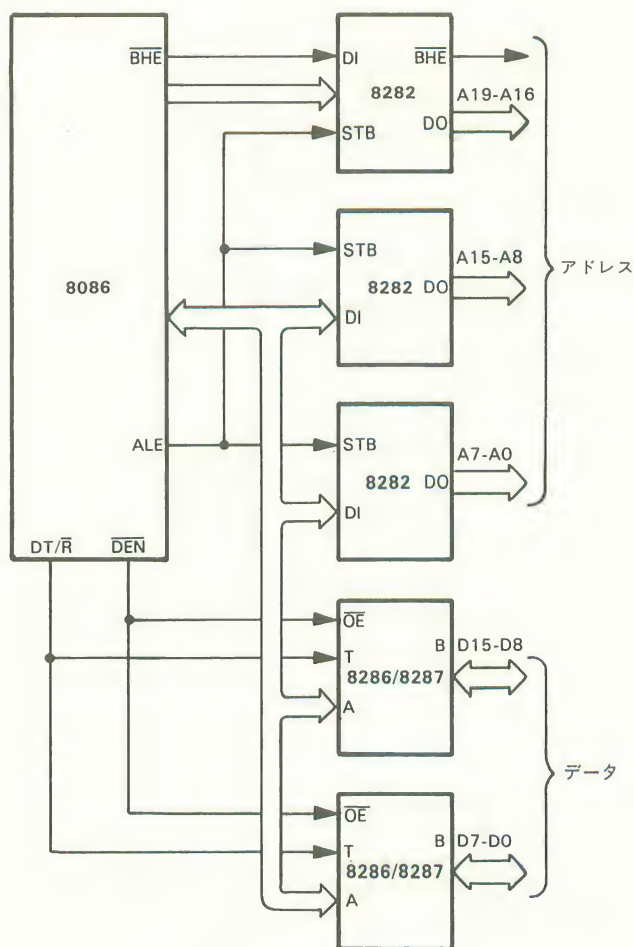


図7-7 バッファを用いたデータ・バス

8086のタイミングは、アドレスがALEによってラッチされて多重化アドレス/データ・バスがフロート状態になるまで、読み出しは有効でないことを保証している。

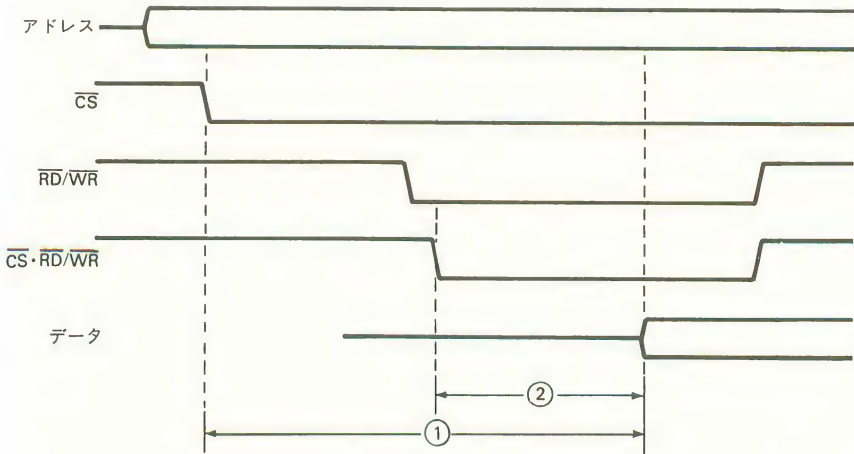


(すべてのインテルのペリフェラル、EPROM製品とRAMは、マイクロプロセッサ向けのものでは、図7-8に示しているように多重化バスに接続させるための出力イネーブルまたはリード入力を備えている)。

出力イネーブルを持たないとき、素子を多重化バスにインターフェイスさせるために、いくつかの方法が存在する。しかし、素子のチップ・セレクトが次のようにコマンドで外部から制御されるならば、各々に制約または制限が生じる。



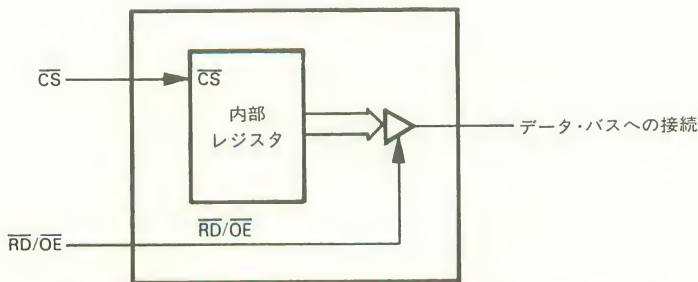
読み込みと書き込みで制御されるチップ・セレクトをもつ図7-8を考える。上にした外部制御が用いられると、2つの問題が生じる。第1に、チップ・セレクトのアクセス・タイムは読み込みのアクセス・タイムにまで落ち、最大のシステム性能（ウェイト・ステートがない）が達せられることを仮定すると、他の不必要に高速な補助素子を必要とさせる。これを図7-9に示す。



- ① アドレス・デコードで生成された $\overline{CS}$ に対するアクセス・タイム
- ②  $\overline{CS}$ が $\overline{RD}/\overline{WR}$ で制御された場合のアクセス・タイム

図7-9  $\overline{RD}/\overline{WR}$ で制御された $\overline{CS}$

図7-8で、分離した $\overline{CS}$ と $\overline{RD}/\overline{OE}$ の入力を持つ素子では、実際にアクセスは $\overline{CS}$ から内部で開始するが、バスへの出力ドライバは、 $\overline{RD}/\overline{OE}$ までイネーブルとはならない。したがって、アクセス・タイムは $\overline{RD}/\overline{OE}$ からではない。これは次のように図示される。

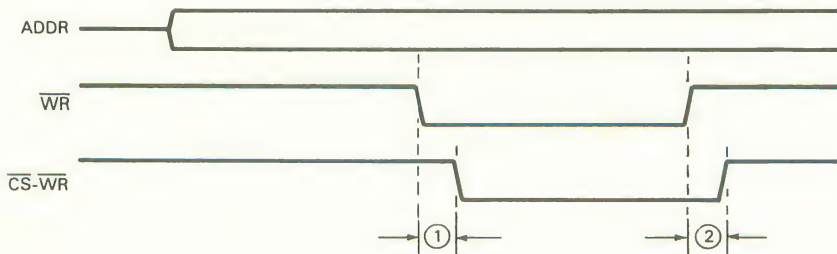


設計者はまた、チップ・セレクトが外部で制御されるとき、素子について、書き込みのセットアップとホールドの時間に対するチップ・セレクトが違反しないことを確かめなければならない。これを図7-10に示す。

別の素子選択の方法も可能であるが、また特殊な制約が課せられる。したがって、出力イネーブルを持つ素子を多重化データ・バスに接続することを勧める。

図7-6に示した多重化データ・バスのもう1つの制限は、特定のAC特性を保証する、8086の2.0mAのドライブ能力とその100pFの容量負荷である。1つのI/O素子につき、20pF、1つのアドレス・ラッチにつき12pF、1つのメモリ素子につき5-12pFの容量負荷を仮定すると、3つの周辺装置と2ないし4個のメモリ素子（各バス・ラインにつき）を合わせたシステムは負荷の制限に非常に近い。

大きいシステムの容量負荷とドライブの必要条件を満たすためには、図7-11に示すようにデータ・バスにバッファを用いなければならない。8286ノンインポートと8287インポートのオクタル・トランシーバは、この必要条件を満たすために、8086ファミリーの一部として提供される。これらは、バス・インターフェイスで32mA、コンポーネント・インターフェイスで10mAを駆動するバッファを持ち、さらに、22ns（8287）または30ns（8286）で、



- ①  $\overline{CS}$ は、書き込みより前に有効ではなく、1または2つのゲート遅延の後にアクティブとなる。  
 ②  $\overline{CS}$ は書き込みの後も1または2つのゲート遅延の間、有効となっている。

図7-10 WRのセットアップとホールドに対する $\overline{CS}$

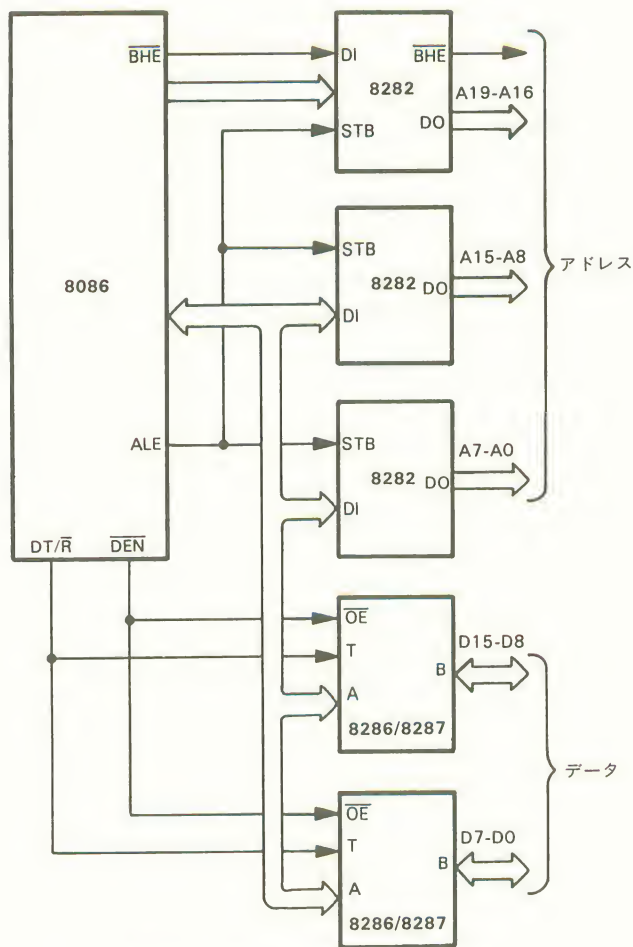


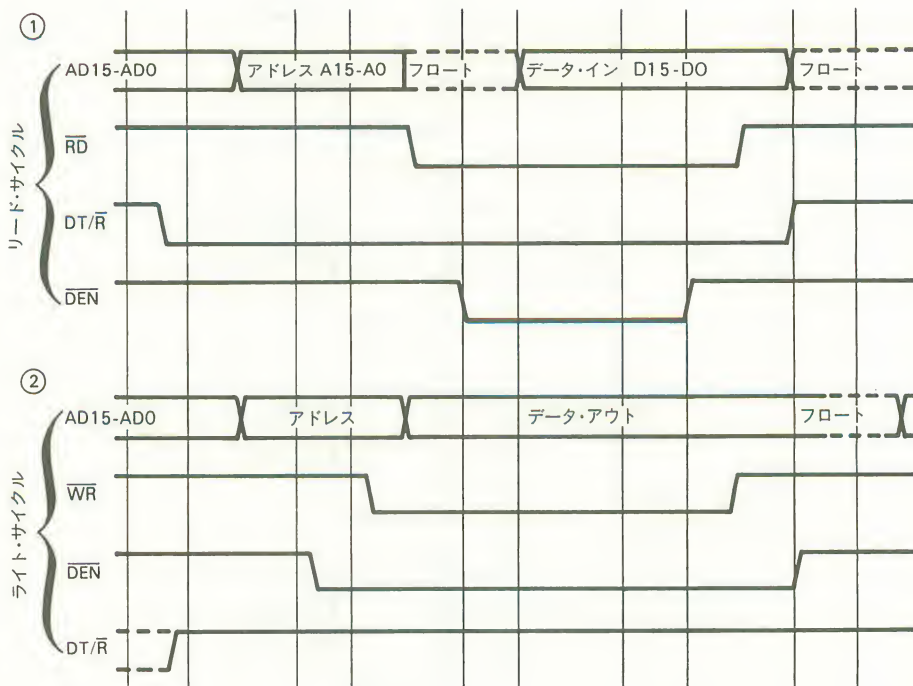
図7-11 バッファを用いたデータ・バス

バス・インターフェイスで300pF、コンポーネント・インターフェイスで100pFの容量負荷のスイッチが可能である。8086システムは、図7-12に示されているように、8286と8287のトランシーバをイネーブルにしてその方向を制御するための、データ・イネーブル( $\overline{\text{DEN}}$ )とデータ・トランスミット/レシーブ( $\text{DT}/\overline{\text{R}}$ )の信号を備えている。

これらの信号は、T1の間、システムから多重化バスの分離を保証し、読み出しと書き込みのバス・サイクルの間、CPUとのバス競合を除去する適当なタイミングを有する。

メモリと周辺の素子がCPUから分離されていても、メモリ/周辺の素子がチップ・セレクトに加えて出力のイネーブル制御を有していなければ、なおシステムにはバス競合が存在する。





- ① 8086が多重化バスをフロート状態とした後に $\overline{DEN}$ はイネーブルになる。
- ②  $\overline{DEN}$ はサイクルの初期にトランシーバをイネーブルにするが、DT/ $\overline{R}$ はトランシーバが受信ではなく送信のモードにあり、CPUに対する駆動を行わないことを保証する。

図7-12 バス・トランシーバの制御

この構成を図7-13に示す。たとえば、1つのチップ・セレクトから他へ遷移する間にバス競合が起き、新しく選択された素子は、以前に選択された素子とそのドライバをディスエーブルにする前に、バスの駆動を開始する。より重要な問題はライト・バス・サイクルの間に生じる。出力がチップ・セレクトだけで制御されている素子は、CPUによってトランシーバを通してデータ出力がライトされると同時に、チップ・セレクトからライト・アクティブまでバスを駆動する。この状態を図7-14に示す。

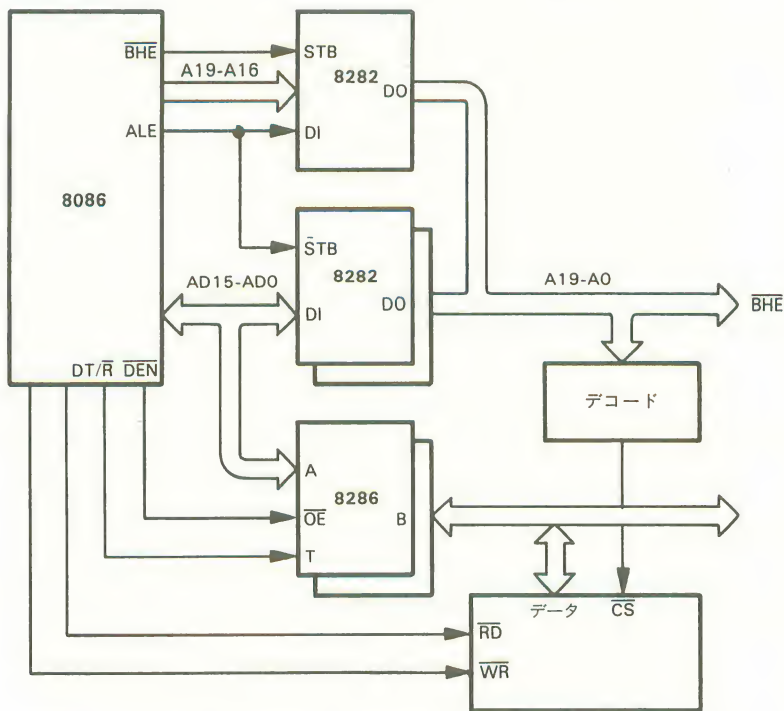


図7-13 システム・バス上のアウトプット・イネーブルを有する素子

多重化バスの周囲の選択タイミング問題に与えられた同じ方法をここで適用できるが、同じ制限を伴う。

第2レベルのバッファによって、システム・バス上で素子から見た全体の負荷を小さくできる。これを図7-15に示す。

一般に、二重バッファは、メモリ・アレイを分離するためにマルチボード・システムで用いられる。しかし二重バッファは、さらにアクセスの遅延を生じ、さらに重要なことに、システム・バスとそれとのインターフェイスを持つ素子への関係で、第2のトランシーバの制御に注意を払わなければならない。第2のトランシーバの制御にはいくつかの方法が利用できる。

図7-16に示す最初の方法は、単にシステム全体に $\overline{DEN}$ と $DT/\overline{R}$ を分配する。 $DT/\overline{R}$ は、第2レベルのトランシーバの適正な方向制御を行なうために、インバートされる。

図7-17に示される第2の方法は、出力イネーブルを持つ素子の制御を与える。

$\overline{RD}$ は通常、周辺装置からシステム・バスへのデータの方向づけに用いられる。図7-17において、ローカル・バス上の素子が選択されたときは常にバッファが選択される。読み込みは同時に素子の出力をイネーブルとしてトランシーバの方向を変えるので、読み込みの間に素子のローカル・バスでバス競合の可能性がある。競合は、読み込みが終結する間でも起きる。

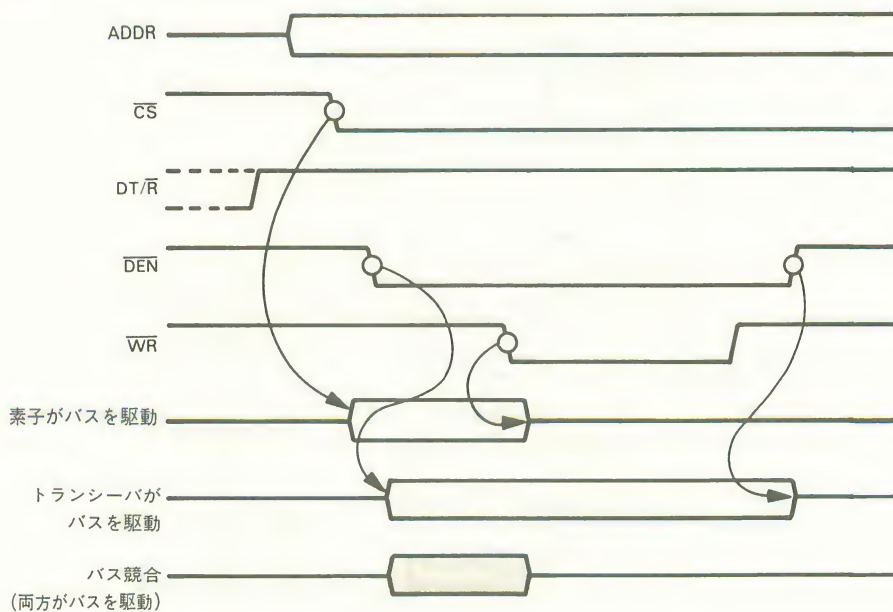


図7-14 アウトプット・イネーブルのない素子のライト期間のシステム・バスにおけるバス競合

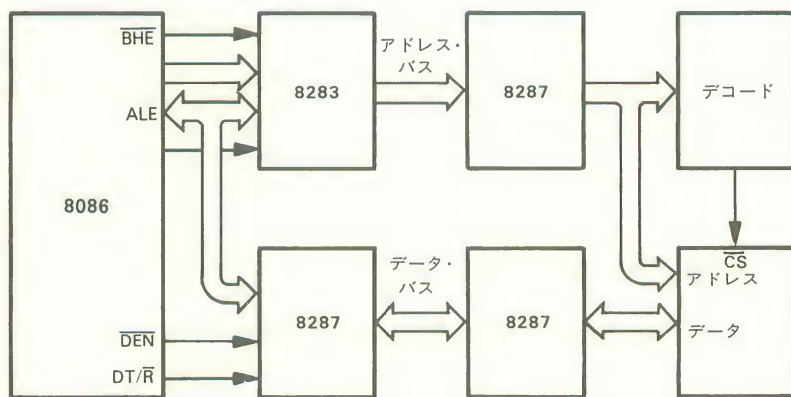
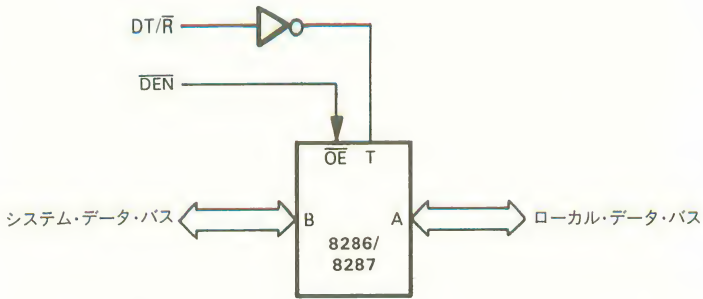
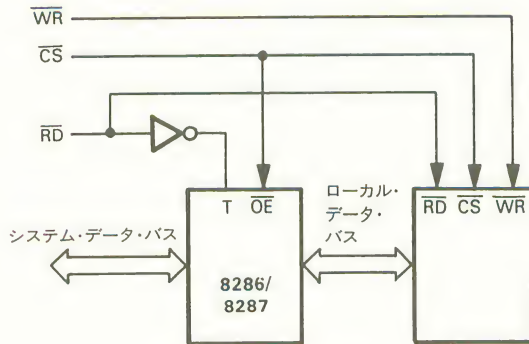
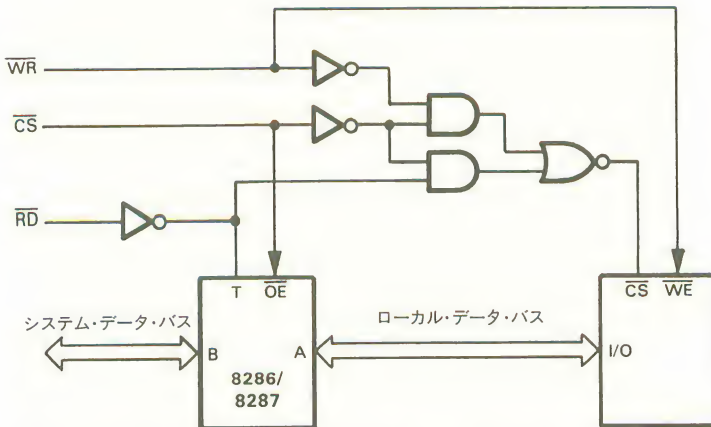


図7-15 全体にバッファを用いたシステム

図7-16  $\overline{DEN}$ と $\overline{DT}/\overline{R}$ によるシステム・トランシーバの制御図7-17  $\overline{OE}$ を有する素子図7-18  $\overline{OE}$ のない素子. 共通または分離の入力/出力制限付きリード・アクセス.  $\overline{WE}$ のホールドとセットアップに対して制限された $\overline{CS}$

出力イネーブルのない素子には、素子のチップ・セレクトが読み込みまたは書き込みで決まるならば、図7-17に示す方法が適用できる。これを図7-18に示す。

読み込み / 書き込みでチップ・セレクトを制御することにより、コマンドが受け付けられる前に素子が第2のトランシーバに対して駆動することを避けられる。この方法での制限を次に示す。

1. 以前述べたように、アクセスは読み込み / 書き込みの時間に制限される。
2. チップ・セレクトは、書き込みのセットアップとホールドの時間に制限される。

出力イネーブルのある素子とない素子に適用される別の方法を図7-19に示す。

$\overline{RD}$  は再び第2のトランシーバの方向を制限するが、コマンドとチップ・セレクトがアクティブとなるまで、イネーブルではない。バス競合の可能性はなお存在するが、トランシーバの方向変更時間に対する出力イネーブルの変動にまで縮小されている。チップ・セレクトからのすべてのアクセス・タイムが利用できる。ただし、データは書き込みより前で

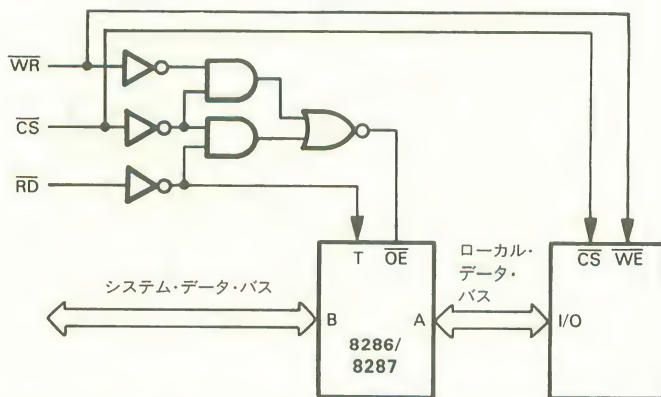


図7-19  $\overline{OE}$  のない素子。共通または分離の入力 / 出力完全リード・アクセス。制限付きのライト・データのセット・アップとホールド

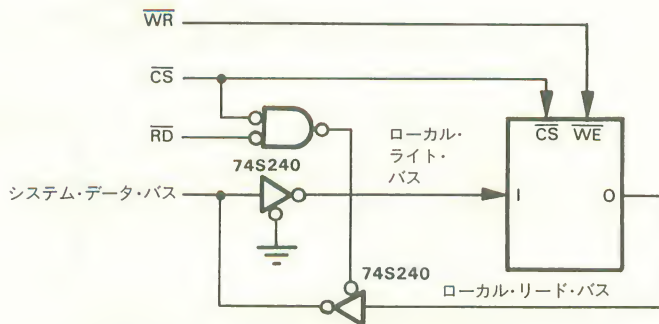


図7-20  $\overline{OE}$  のない素子。分離の入力 / 出力



は、有効でなく、書き込みの後のトランシーバをディスエーブルにするのに必要な遅延量だけ有効として保持されているだけとなる。

最後の1つの方法は、分離した入力と出力を持つ素子のためのものである。図7-20を見られたい。

今までに示した1個のトランシーバではなくて、分離したバスのレシーバとドライバが用いられている。レシーバは常にイネーブルであるが、一方バス・ドライバは、 $\overline{RD}$ とチップ・セレクトによって制御されている。このシステムでバス競合の起きる可能性は、チップ選択が変化する間に、リード・バスの各ライン上の複数の素子がイネーブルとなってディスエーブルとなるとときだけにある。

8086のインターフェイスについてはこの節を通して、多重化バスは“ローカル”なCPUバス、分離されたアドレスとバッファを用いたデータのバスは“システム・バス”と考えている。

#### 7.2.4 8086のエグゼキューション・ユニットとバス・インターフェイス・ユニット

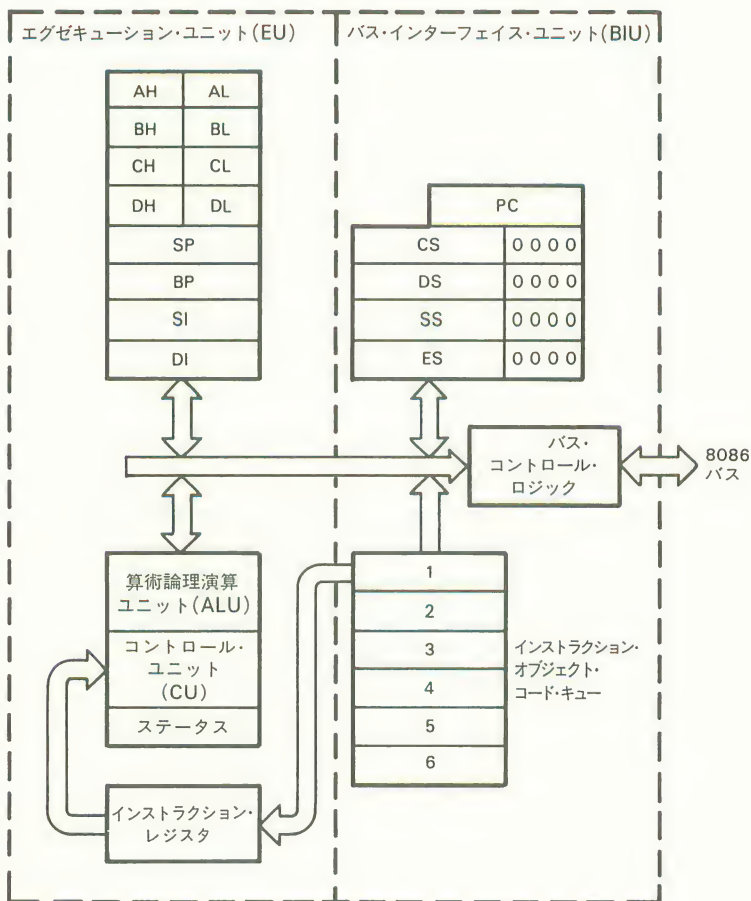
8086の命令実行のタイミングを調べるときに、理解すべき最も重要な概念は、8086バス・コントロール・ロジックは8086命令実行ロジックと分離されているという事実である。すなわち、8086にはエグゼキューション・ユニット (EU) とバス・インターフェイス・ユニット (BIU) がある。

EUには、データとアドレスのレジスタ、算術論理演算ユニット、さらにコントロール・ユニットが含まれている。BIUには、バス・インターフェイス・ロジック、セグメント・レジスタ、メモリ・アドレッシング・ロジック、さらに6バイトの命令オブジェクト・コード・キューが含まれる。これは次ページに示す図のようになる。

エグゼキューション・ユニット (EU) とバス・インターフェイス・ユニット (BIU) とは非同期に動作する。EUが新しい命令を実行できるときは常に、BIUの命令キューから命令のオブジェクト・コードをフェッチして、バス・サイクルとは何の関係もないある数のクロック期間で命令を実行する。もし命令オブジェクト・コード・キューが空ならば、BIUは命令フェッチのマシン・サイクルを実行し、CPUは命令オブジェクト・コードがフェッチされるのを待つ。しかし、まもなく明らかになる理由から、キューはめったに空とはならない。したがって、EUは、命令フェッチが行なわれる間、一般に待たなければならないことはない。

メモリまたはI/O素子に、命令の実行中にアクセスしなければならないならば、EUはBIUにその必要を通知する。BIUは、EUの要求に応じて、適当な外部アクセスのマシン・サイクルを実行する。

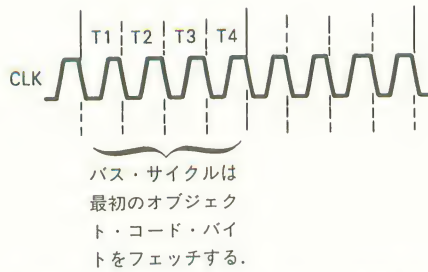
BIUはそれ自体、EUと独立であり、6バイトのキューを命令オブジェクト・コードで満たしておくことを試みる。もしこの6バイトの2ないしそれ以上が空ならば、EUはバス・アクセス保留の有効な要求がなければ、BIUは命令フェッチのマシン・サイクルを実行する。BIUが命令フェッチのマシン・サイクルの途中にある間に、EUがバス・アクセスの要求を出すと、BIUはEUのバス・アクセスの要求を引き受ける前に、命令



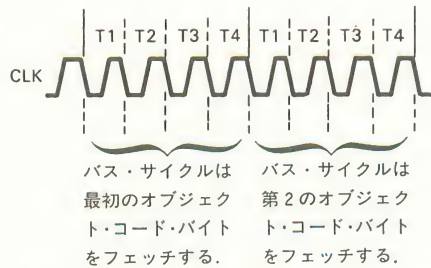
フェッチのマシン・サイクルを完了する。

### 7.2.5 8086命令キュー

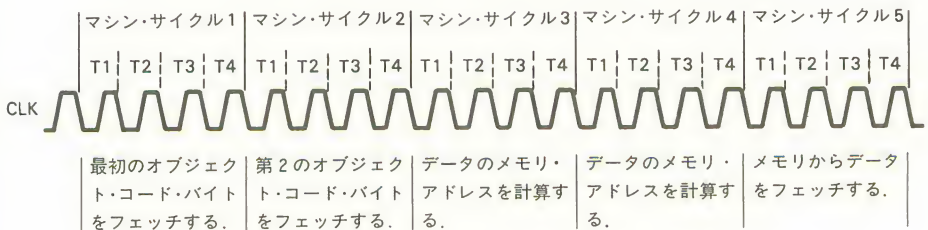
命令が実行されるときに何が起きるかについて考える。簡単な場合から始めて、バス・インターフェイス内の命令オブジェクト・コード・キューは空とする。したがって、EUが命令を要求すると、BIUは命令の第1のバイトをフェッチするためのバス・サイクルを実行する。

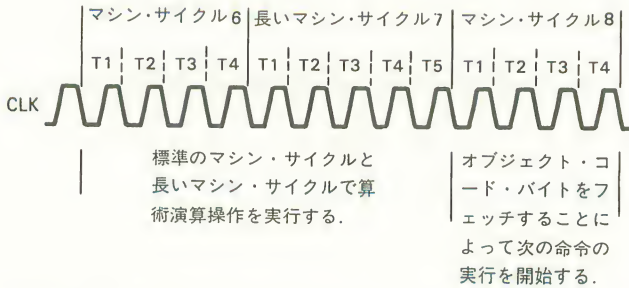


この特定の命令は、2 バイトのオブジェクト・コードを必要とすると仮定する。事情を簡単にして、次の命令バイトをフェッチするために直ちに実行されるもう 1 つのバス・サイクルを示す。

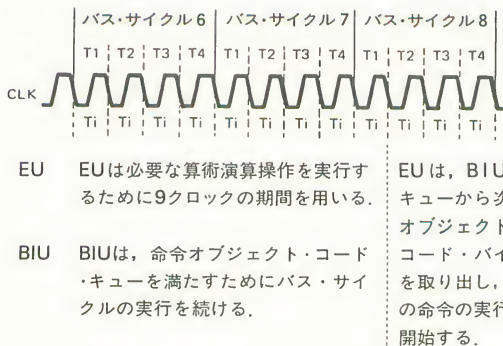
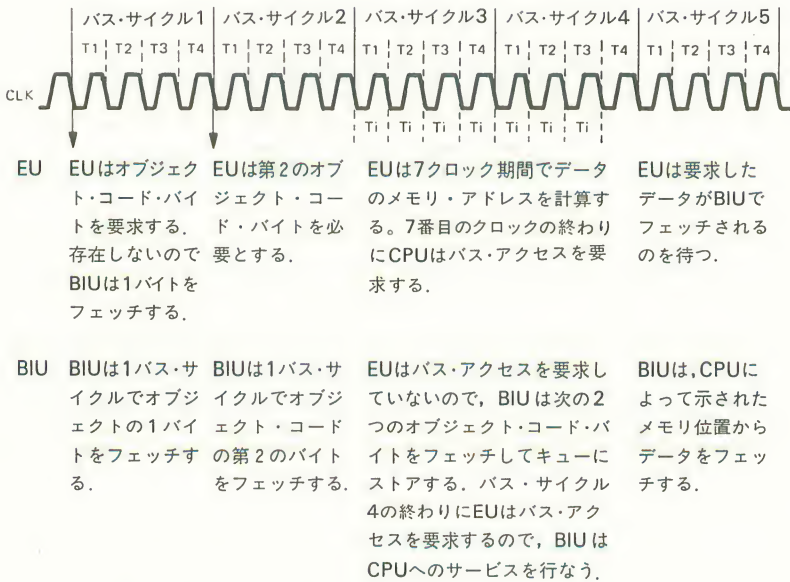


この命令は、メモリから 1 ワードのデータを読み込み、このデータを用いて算術演算操作を行なうと仮定する。命令は、アクセスすべきデータのメモリ位置の有効アドレスを計算するために、いくつかのクロックの期間を必要とする（7クロックの期間を要すると仮定する）。さらにまた、算術演算操作を行なうためにいくつかのクロックの期間が必要となる（9クロックの期間を仮定する）。通常のマイクロプロセッサでは、この命令は、次の一連のマシン・サイクルとして実行される。

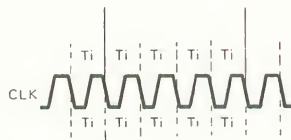
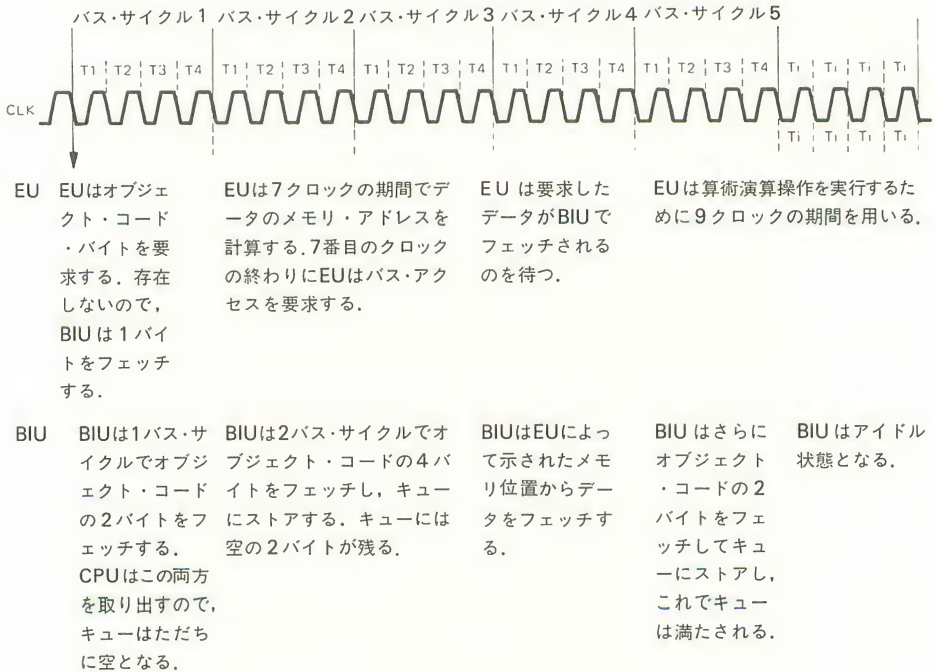




しかし、非同期のCPUとバス・コントロール・ユニットのロジックを持つ8086は、上に示した命令を実行するために、クロック期間を次のように用いる。



ところで、記憶されているように、データ・アドレスが偶数バイト境界にあれば、8086は16ビットのインクリメントを行なってデータをフェッチするので、上に示した図は正確でない。また、BIUは、少なくともキューに2バイトの空があるときに限り、命令のバイトをフェッチしてキューにロードする。すべてのデータが偶数バイト境界にあると仮定すると、タイミングは次のようになる。



- EU EUは命令の実行を終わり、次の命令を実行するためにキューから1バイトのオブジェクト・コードをフェッチする。
- BIU 1バイトだけキューが空なのでBIUはアイドル状態のままである。



8086バス・サイクルのタイミングについて注意すべきいくつかの重要な点がある。バス・サイクルは、BIUの現象である。

EUロジックに関する限り、バス・サイクルは存在しない。EUは、命令を実行している活動期間と、命令のオブジェクト・コードあるいはBIUがバス・サイクルによって処理するデータを待っている不活動期間を経験する。EUの活動の期間は、一連のクロック期間で時間が定まる。EUは、クロック期間をマシン・サイクルにまとめることを試みたりはしないし、クロック期間が特定の数値の組合せで生じることもない。

BIUに関する限り、データが8086との間で伝送されるべき場合にだけ、クロック期間にはバス・サイクルにまとめられる。第1のプライオリティは、EUから出されるバス・アクセスの要求に与えられている。EUがバス・アクセスを要求しなければ、BIUはキューが満たされるまで、命令フェッチのバス・サイクルを実行する。以下は、BIUが命令フェッチのバス・サイクルを実行するために、前もって必要な事柄である。

1. バス・サイクルを初期化するクロック期間は、初期化の期間でなければアイドルのクロック期間である。
2. EUは、保留の有効なバス・アクセス要求を持たない。
3. 少なくともキューに2バイトの空きがある。

キューが満たされていれば、BIUはバス・サイクルの実行を中止し、前に示したように、一連のアイドル・クロック期間が生じる。

CPUはバス・アクセスを待たなければならないことに注意。前に示した図で、CPUは、データのメモリ・アドレスを計算するために、7クロックの期間を要する。7番目のクロック期間の終わりに、EUはBIUに対してバス・アクセスの要求を出す。しかしこの時点でBIUは、命令フェッチのバス・サイクルの間、分離した状態にある。BIUは、命令フェッチのバス・サイクルを終了して、EUのバス・アクセス要求を引き受ける。

前に示した最後の図で、新しい命令の実行開始にバス・サイクルは伴わない。実行される次の命令は1バイトのオブジェクト・コードを持つことを仮定している。このオブジェクト・コード・バイトはキューの前からフェッチされ、キューは、丁度1バイトの空きを持つ。キューからコードが取り出されるので、命令オブジェクト・コードをフェッチするためのバス・サイクルは実行されない。続いて、1バイトだけの空きが存在するので、BIUは命令フェッチのバス・サイクルを実行しない。BIUが命令フェッチのバス・サイクルを実行するには、キューに少なくとも、2バイトの空きが存在しなければならない。8086の命令フェッチのキュー処理についての前述の解説を基礎に、8086が本質的に命令フェッチの時間を省略していることが見られる。BIUが命令オブジェクト・コードをフェッチする間に、EUがウエートしなければならないのは、条件付き分岐命令によってキューの系列からの分岐が実行されたとき、あるいは(何かの理由で)命令実行に伴うメモリ・アクセスがあまりにも頻繁で、命令フェッチのバス・サイクルを挿入するのに十分なアイドル・クロック期間がBIUにないときに限る。

システム・デザインに影響する、キューに関するその他の情報については、レディの実現とタイミングの節の最後の所を参照のこと。

# 第8章

## 8086の基本デザイン

### 8.1 動作モード

8086は、様々で著しく異なるアプリケーションでも容易に動作するように、構成される。7章で述べた  $\overline{MN}/\overline{MX}$  入力は、“ミニマム・モード”と“マキシマム・モード”として区別される出力の2つの異なる組として、8086を機能させる接続による選択となっている。

ここで2つのモードについてさらに詳しく検討する。

#### 8.1.1 ミニマム・モード

ミニマム・モードの8086は、 $\overline{MN}/\overline{MX}$  ピンを  $V_{cc}$  に接続する。ミニマム・モードは、1または2枚のボードのシングルCPUシステムで用いる。図8-1にミニマム・モードの8086を示す。

ミニマム・モードでは、8086は1メガバイトのメモリ領域と64キロバイトのI/O領域の全体をアドレス指定できる。データ・バスは16ビットの幅を持つ。8086は直接に、バス・コントロール( $\overline{DT}/\overline{R}$ ,  $\overline{DEN}$ ,  $\overline{ALE}$ ,  $\overline{M}/\overline{IO}$ ,  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{INTA}$ ) を供給する。既存のDMAコントローラと互換性のある、簡単なCPUの権利獲得の機構は、 $\overline{HOLD}$  と  $\overline{HLDA}$  の信号によって可能となっている。

#### 8.1.2 マキシマム・モード

図8-2に示されているマキシマム・モードは、 $\overline{MN}/\overline{MX}$  ピンをグラウンドに接続する。マキシマム・モードは、マルチプロセッサとコープロセッサ構成で用いられる。

マキシマム・モードにおいて、8288バス・コントローラは入力として8086からコントロール信号を受け取る。この入力は、コントロールの出力信号を生成するために、8288によってデコードされる。8086の他のコントロールの信号も、外部ロジックに情報を与えるた

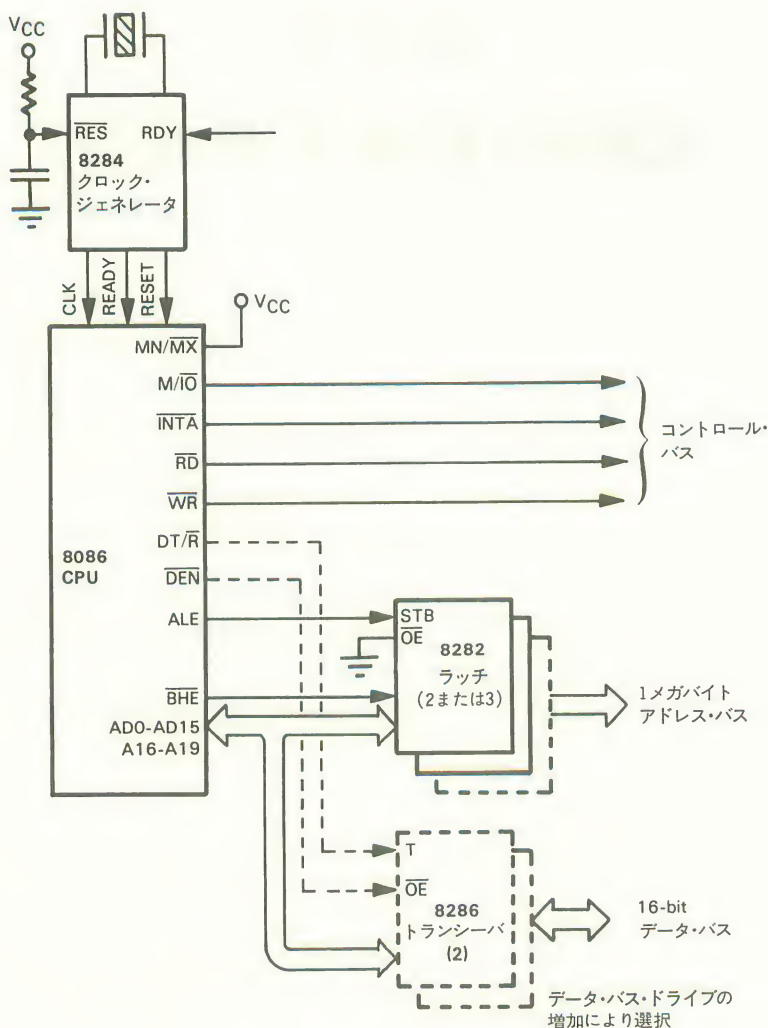


図8-1 ミニマム・モード8086

めに変更される。特に、8086の出力信号を次のように再定義する。

1. キュー・ステータスは、QS 0と QS1に出力される。これにより、外部素子、たとえば ICE/86 あるいは特定の命令セット拡張のコプロセッサが、CPUの命令実行に追従することが可能となる。
2. システムのコントロールと構成のオプションは、バス・サイクル・ステータスの出力、 $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  によって拡張される。これらの出力は、8288バス・コントローラ、

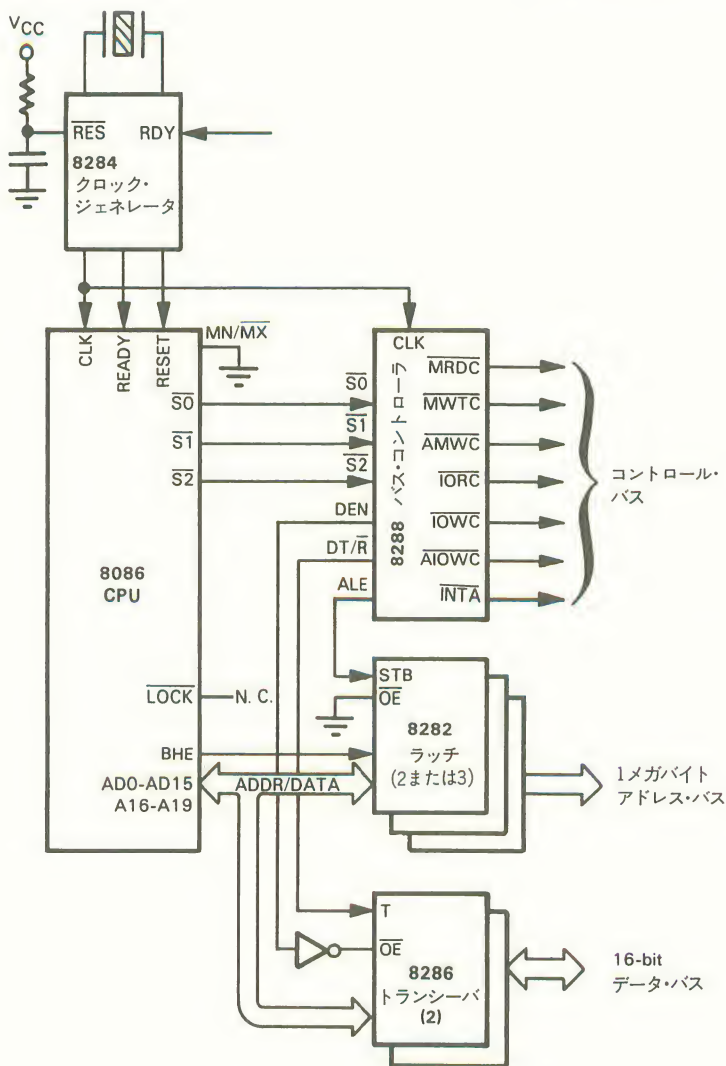


図8-2 マキシマム・モード8086

8289バス・アービタ\* そして類似の外部素子によって用いられる。

3. マルチプロセッサ・システムにおいて共有資源に対するアクセスの制御は、バス・ロックの機構によってサポートされている。
4. プロセッサ獲得の2つの優先順位のあるレベル ( $\overline{RQ/GT0}$ ,  $\overline{RQ/GT1}$ )によって、システム・バス・インターフェイスを共有する8086のローカル・バス上に、複数プロ

\* arbiter: (権利)判定用素子 (訳者注)。

セッサの存在が可能となる。

次に、これらの拡張された能力がどのように用いられるかを検討する。

キュー・ステータスは、内部のキューからどのような情報が移動されているか、コントロールが移されたことによって、いつキューがリセットされるかを示す。

表8-1に、キュー・ステータスの意味の要約を示す。

表8-1 キュー・ステータスの出力

| QS1                                      | QS0 | 意 味                |
|------------------------------------------|-----|--------------------|
| 0                                        | 0   | ノー・オペレーション         |
| 0                                        | 1   | キューからのオペコードの第1のバイト |
| 1                                        | 0   | キューが空              |
| 1                                        | 1   | キューからの後続のバイト       |
| キューの操作が行なわれた後のCLKサイクルの間<br>キュー・ステータスは有効。 |     |                    |

図8-3に示されているものと同様のロジックを用いれば、8086のキュー・ステータスを追跡することができる。S0, S1,

S2がそれぞれ1, 0, 0のとき、命令フェッチが実行されている。QS0とQS1は、命令が8086のキューあるいは外部メモリのどちらからフェッチされているかを示す。外部メモリのアクセスに対して、A0とBHEはワードあるいはバイトのアクセスを表わす。このロジックは多くの方法で用いることができる。以下の例を考える。

ICE/86は、特定のメモリ位置にストアされた命令の実行を追跡できる。キューを追跡するためにICE/86によって用いられる回路の例を図8-3に示す。

第1のアップダウン・カウンタはキューの深さを追跡し、第2のアップダウン・カウンタは釣り合うキューの深さを補える。第2のカウンタは、キューが空になるまで、キューからさらにフェッチするに従って減少するか、釣り合うアドレスの実行を表示して、カウントは0になる。第1のカウンタは、キューからのフェッチ(QS0=1)によって減少し、キューにオブジェクト・コード・バイトがストアされると増加する。外部メモリからの通常の命令フェッチはキューに2バイトを移動し、カウンタは2つのクロックの増加(T201とT301)となることに注意。バスの上位(A0がハイでBHEがロー)から1バイトがロードされたときは、カウンタは1回増加する。EUはBIUに同期していないので、キューからのフェッチはキューへの移動と同時に起きる。第1のカウンタのENP入力を駆動するエクスクルーシブORゲートにより、同時操作を互いにキャンセルしてキューの深さを変更しないようにしている。

8086のキューは、ESCAPE命令の実行を検出するためにコープロセッサによって追跡される。ESCAPE命令は、コープロセッサにある特殊なタスクの実行を指令する。

表8-2に、ステータス・ラインS0, S1, S2の意味を定義する。前に述べたように、これらのステータス・ラインは、いつバス・サイクルを初期化するか、どのようなタイプのコマンドを発行するか、そしていつバス・サイクルを終了するかを、8288に通知する。8288は、CPUクロック期間の開始に

表8-2 ステータス・ラインの出力

| S2 | S1 | S0 | 意 味           |
|----|----|----|---------------|
| 0  | 0  | 0  | インタラプト・アクノリッジ |
| 0  | 0  | 1  | リードI/Oポート     |
| 0  | 1  | 0  | ライトI/Oポート     |
| 0  | 1  | 1  | ホルト           |
| 1  | 0  | 0  | コード・アクセス      |
| 1  | 0  | 1  | リード・メモリ       |
| 1  | 1  | 0  | ライト・メモリ       |
| 1  | 1  | 1  | パッシブ          |





ステータス・ラインをサンプルする。バス・サイクルの初めに、CPUはステータス・ラインをパッシブ状態 ( $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  がすべてハイ) から可能な7つのアクティブ状態の1つに変える。

新しいバス・サイクルに対して、以前のバス・サイクル期間  $T4$  のクロックの立ち上がりエッジ、または現在バスが動作していなければ  $T1$  のアイドル・サイクルの間に、8086は  $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  の状態を変える。8288は、クロックのハイからローへの遷移でステータス・ラインをサンプルすることによって、ステータスの変化を検出する。

8288は、適当なバッファ方向のコントロールを伴い、ハイの  $\overline{ALE}$  パルスを生成することによってバス・サイクルを開始する。これは、ステータス変更の検出直後のクロック期間に生じる。バス・トランシーバと指定された操作は、次のクロックの期間、動作可能となる。ステータスがパッシブ状態に戻ると、8288は動作を終了する。タイミングを 図8-4 に示す。

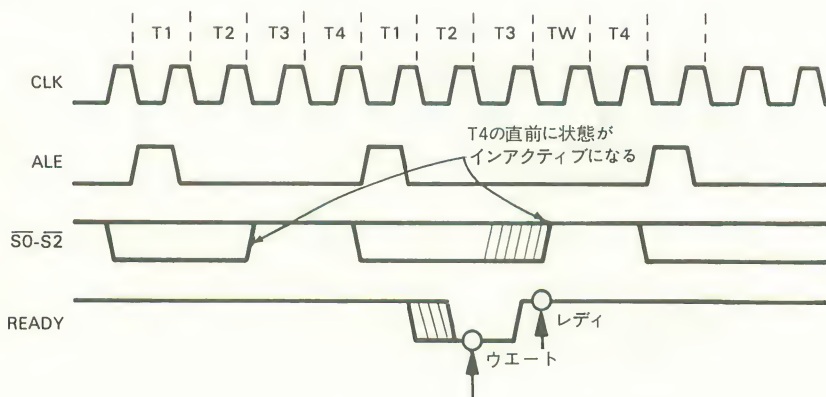


図8-4 ステータス・ラインのアクティブ化と終結

8086は、ウエート状態の間、 $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  のレベルを維持する。前に述べたように、8284クロック・ジェネレータにローの  $\overline{RDY}$  信号を入力する外部ロジックによって、ウエート状態が引き起こされる。クロック・ジェネレータはハイの  $\overline{READY}$  信号を出力するが、これは  $\overline{CLK}$  との同期がとられて8086に伝達される。

8086はウエート状態の間、 $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  のレベルを維持するので、任意のクロック期間にわたるウエート状態に対して、8288はアクティブなバス・コントロールを保持する。バスの動作をモニタし、他のプロセッサがローカル・バスのコントロールを獲得したならば8288をコントロールするために、8086のローカル・バス上の他のプロセッサによっても、ステータス・ラインは用いられる。

8288は、バス・コントロール信号  $\overline{DEN}$ ,  $\overline{DT/R}$ ,  $\overline{ALE}$  とコントロール信号  $\overline{INTA}$ ,  $\overline{MRDC}$ ,  $\overline{IORC}$ ,  $\overline{MWTC}$ ,  $\overline{IOWC}$ ,  $\overline{AIOWC}$  を供給する。コントロール信号は、Intel MULTIBUS との互換性のために、メモリとI/Oに対するリード/ライトの操作を分離する。

周辺装置やスタティックRAMによってしばしば要求される広いライト・パルス幅を供

給するために、通常のライト・コントロール信号より1クロック期間速い、アドバンスド・ライト・コントロール信号が使用できる。通常のライト・コントロール信号により、ライト・コントロール・パルスの立上りエッジでデータのストロブを行なうダイナミックRAMメモリとI/O素子に適應するために、ライト・パルスの前にデータが設定される。アドバンスド・ライト・コントロール信号は、コントロール信号の立上りエッジより前で、データが有効であることは保証していない。

マキシマム・モードにおけるDEN信号は、ミニマム・モードと比較すると反転している。これはDENを、他の信号、特にインタラプト・コントロールとロジック・ゲートで組み合わせることを容易にしている。図8-5に、ミニマム・モードとマキシマム・モードのバス転送コントロール信号のタイミングの比較を示す。

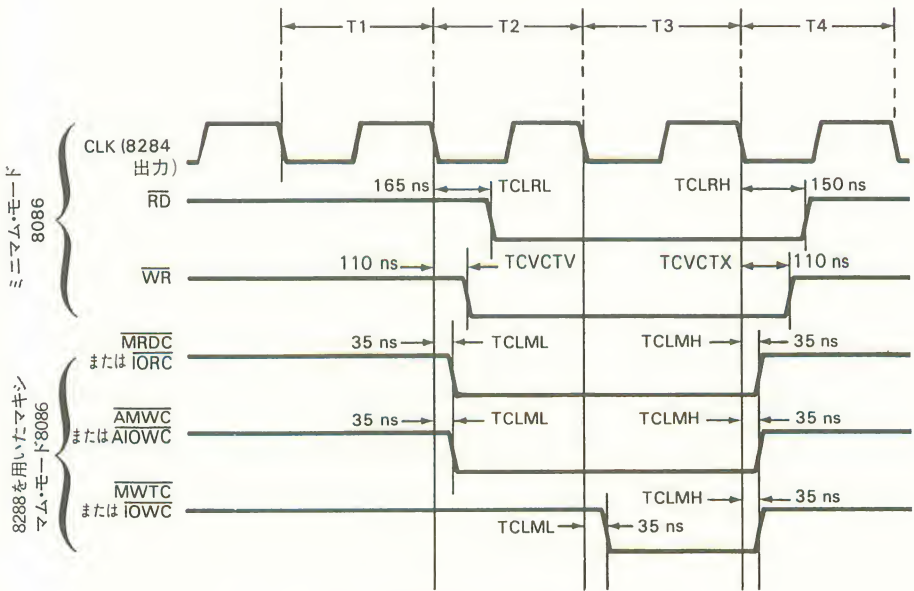


図8-5 ミニマムとマキシマムのバス転送タイミング

マキシマム・モードは、多重処理構造と、規模の大きいシングルCPUデザイン（マルチバス・システムあるいは、2つまたはそれ以上のPCボードを含むシステム）のために設計されている。8288はバイポーラ素子である。したがって、コントロール信号の32 mAの駆動能力と、タイミング・パラメータの許容誤差と最悪の場合の遅延から、ミニマム・モードの8086に比較して規模の大きい良い性能のシステムが得られる。

8086から取り除かれた機能に加えて、8288は、マルチプロセッサ構成と8086のローカル・バスの周辺素子をサポートするための付加的な接続による選択とコントロールを与える。この能力は、メモリまたはI/Oを含む資源を共有あるいはローカルなものとして割り当てることを可能にする。共有の資源は、Multibus システム・バスで利用できる。ローカルな

資源は、そのローカル・バス上の8086によってだけアクセスできる。この技法によって、Multibus システム・バスに対するアクセスの競合が減少し、マルチCPUシステムの性能が改善される。特殊な構成については後の章で述べる。

8086 マキシマム・モードの  $\overline{\text{LOCK}}$  出力は、共有資源に対するアクセスのコントロールに役立つ。 $\overline{\text{LOCK}}$  の出力は、8086 が LOCK プレフィックス命令を実行すると、アクティブになる。 $\overline{\text{LOCK}}$  の出力は、LOCK プレフィックスの実行に続く最初のクロックの期間でローになり、LOCK プレフィックスに続く命令の最後の命令実行クロック期間中と次の命令実行の最初のクロック期間でローになっている。 $\overline{\text{LOCK}}$  の信号は、すべてのマイクロプロセッサのシステム・バス判定ロジックの一部でなければならない。

通常のマルチプロセッサ・システムの動作の間、共有のシステム・バス・アクセスの優先権は、1 サイクルごとを基礎に判定回路によって決定される。ある8086にシステム・バスを通してデータを伝送する必要があると、バス・アクセスを要求する。特定のシステム・バス判定方法によって決定されて、この8086が優先権を獲得すると、システム・バスのコントロールを得てバス・サイクルを実行し、次に、システム・バスのコントロールを保持するか、自主的にシステム・バスを解放するか、あるいは優先権を失うことによって強制的にシステム・バスから切り離される。

LOCK機構は、自主的であってもなくても、8086がシステム・バスのコントロールを失うことを防止する。これにより、他のCPUによる干渉と可能性のあるデータの破壊を受けずに、8086が複数のバス・サイクルの命令を実行する能力を保証している。LOCK の出力の動作を図8-6に示す。

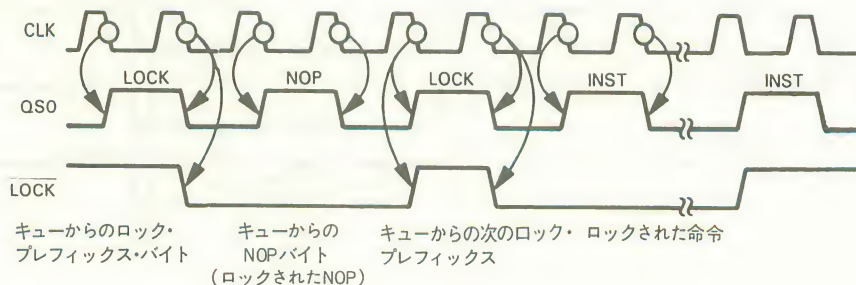


図8-6 LOCK の動作

$\overline{\text{LOCK}}$  の出力は、離れてロックされた命令の間ではインアクティブになることに注意。また、LOCK プレフィックスは実行時間に2つのクロック期間を付加している。

キュー・ステータスは以前のクロック期間のキューの動作を反映しているので、 $\overline{\text{LOCK}}$  の出力は、実際に次の（ロックされた）命令の開始に一致してアクティブとなり、ロックされた命令の実行に続く1クロックの間、アクティブとなる。

LOCK プレフィックスに続く命令のオブジェクト・コードがキューになければ、示したように命令のオブジェクト・コードが外部メモリからフェッチされる間、 $\overline{\text{LOCK}}$  の出力はローとなる。



バス・インターフェイス・ユニット(BIU)は、ロックされた命令の実行期間も命令フェッチ・サイクルを実行する。LOCKは単に、命令実行の間、1つの8086がシステム・バスのコントロールを保持することを保証するだけで、このCPUがこのロックの期間に行なうことの可能なバス動作のタイプを決して制限するものではない。

LOCK機構は、TESTとSETのハンドシェイク・シーケンスで一般に用いられる。このシーケンスの間、8086は共有メモリ位置から読み出しを行ない、その位置にデータを戻す。他のどのCPUも、リード操作のTESTとライト操作のSETの間は、このメモリ位置を参照することができない。8086はこの動作を次のようにロックされた交換命令で行なう。

```
LOCK  XCHG  reg, memory ; reg は8086の任意の
                        ; レジスタ, memory は信号
                        ; のアドレス
```

マルチプロセッサ・システムにおいて、LOCKの興味ある別の用途は、1つのCPUのメッセージ・バッファから他へ高速のメッセージ転送を可能とする、ロックされたブロックの移動である。

ロックされた命令の期間、 $\overline{\text{RQ}}/\overline{\text{GT}}$ ラインによって発生するプロセッサ獲得の要求は記録されるが、ロックされた命令の完了までは受け付けられない。

LOCKプレフィックスは、割り込みに対して直接的影響を持たない。一般には、プレフィックスのバイトは、それが先行する命令の拡張と考えられる。したがって、プレフィックスの実行中に発生した割り込みは、プレフィックスに続く命令の完了まで受け付けられない(割り込みは有効であると仮定)。

複数のプレフィックスのバイトが1つの命令に先行できることに注意。繰返しのプレフィックス(REP)は、続く命令の実行の度に割り込みで中断できる。このことは、REPがLOCKプレフィックスと結合されても成り立つ。したがって、割り込みはブロック移動あるいは繰返しのストリング操作で、ロックの影響を受けない。この動作と複数プレフィックスのストリング操作についてのこれ以上の情報は、この章の後の8086インタラプト構造を取り扱う節で述べる。

優先づけられたプロセッサ獲得の付加的レベルについては、この章の後で詳細に述べる。

## 8.2 クロックの発生

8086は、ローが $-0.5$ から $+0.6$ までとハイが $+3.9$ から $V_{cc}+1.0$ の電圧の間の、速い立上りと立下りの時間(最大10ナノ秒)を有するクロック信号を必要とする。8086の最大クロック周波数は5 MHzである。8086の設計はダイナミック・セルを取り入れているので、2 MHzの最低周波数が必要とされる。最低周波数条件から、CPUのシングル・ステップあるいはサイクリングは、クロックを無効にすることによっては果たされない。CPUクロックのタイミングと電圧の必要条件を図8-7に示す。

一般に、最大値以下の周波数に対して、CPUのクロックは、8086のデータ・シートに述べられている周波数依存のパルス幅制限を満たす必要はない。指定されている値は満た



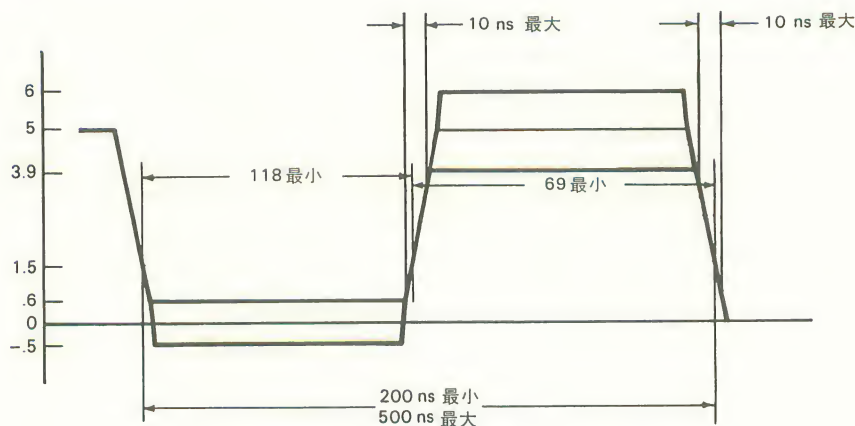


図8-7 8086CPUのタイミングと電圧の必要条件

されるべき最小値を反映しているだけで、最大クロック周波数の立場から述べられている。クロック周波数がCPUの最大周波数に近づくと、CPUの最小クロックのローとハイの時間の仕様を満たすために、クロックは33%のデューティ・サイクルに適合しなければならない。

必要な電圧レベルと遷移時間を有する最適な33%デューティ・サイクルのクロックは、図8-8に示すように、8284クロック・ジェネレータで得ることができる。

外部周波数源あるいは直列共振クリスタルで8284が駆動される。選ばれたソースは、必要なCPU周波数の3倍（3×）で発振しなければならない。クロック発生の周波数源として8284のクリスタル入力を選ぶためには、8284への $F/\overline{C}$ 入力はグラウンドに接続されなければならない。この接続による選択で、クロック・ジェネレータのソースとして、クリスタルまたは外部周波数の入力を選ぶことができる。8284はオーバートーン・モードのクリスタルに適應するタンク回路の入力を有しているが、より正確な（そして安定な）周波数発生のために基本モードのクリスタルを推奨する。8284に用いるためにクリスタルを選ぶときは、直列抵抗はできるだけ低いことが必要である。

他の回路要素は動作周波数を共振から変化させる傾向があるので、動作インピーダンスは指定された直列抵抗よりも一般に高くなっている。オシレータの帰還回路の減衰がループ利得を1以下に下げると、オシレータは動作しなくなる。

電圧と温度の変動に対してオシレータの安定性を最も良くするためには、X1とX2へのクリスタルの接続に510Ωの抵抗で接地することを勧める。インテル

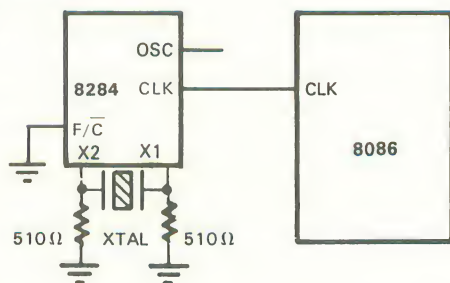


図8-8 8284を用いたCLKの供給

のマイクロプロセッサのクリスタルを供給している多くのベンダーの2社を表8-3に、関連のあるいくつかの周波数のクリスタル部品番号と共に示す。

表8-3 クリスタル・ベンダー

| f          | パラレル / シリーズ | Crystek <sup>(1)</sup><br>Corp. | CTS Knight <sup>(2)</sup><br>Inc. |
|------------|-------------|---------------------------------|-----------------------------------|
| 3.6 MHz    | P           | **                              | **                                |
| 5.185 MHz  | S           | CY8A                            | **                                |
| 6.0 MHz    | P           | **                              | MP060                             |
| 6.144 MHz  | P           | **                              | MP061                             |
| 6.25MHz    | P           | **                              | MP062                             |
| 10.0 MHz   | P           | **                              | MP10A                             |
| 15.0 MHz   | S           | CY15A                           | MP150                             |
| 18.432 MHz | S           | CY19B*                          | MP184*                            |
| 24.0 MHz   | S           | **                              | MP240                             |
| 25.0 MHz   | S           | **                              | MP250                             |
| 27.0 MHz   | S (オーバートーン) | CY27A                           | MP270                             |

\* このアプリケーションに対しては Intel もクリスタル番号8801を供給している。  
 \*\* 適当な仕様でベンダーに連絡のこと。  
 注) 1.住所: 1000 Crystal Drive, Fort Meyers, Florida 33901  
 2.住所: 400 Reimann Ave., Sandwich, Illinois

高精度の周波数源、外部可変の周波数源、あるいは複数の8284を駆動する共通のソースが必要な場合、図8-9に示すように、1 キロオームを通して  $F/\overline{C}$  入力を +5 ボルトに接続することによって、8284のエクスターナル・フリーケンシー・インプット (EFI) を選ぶことができる。

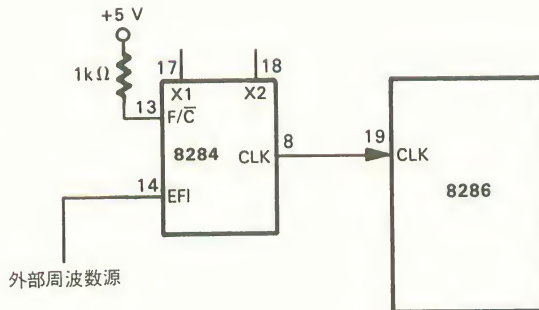


図8-9 外部周波数源の利用

外部周波数源は、TTLと互換性があり、50%のデューティ・サイクルを持ち、必要なCPU動作周波数の3倍で発振しなければならない。8284が受け付けられる最大EFI周波数は、クロックのローとハイの時間が最小13nsで、24MHzよりも少し高い。最小EFI周波数は指定されていないが、CPUの最小クロック速度に違反してはならない。

システムに分散する8284を駆動するために共通の周波数を用いるならば、各8284はソースからの個別のラインで駆動されなければならない。システムのノイズを最小にするため、

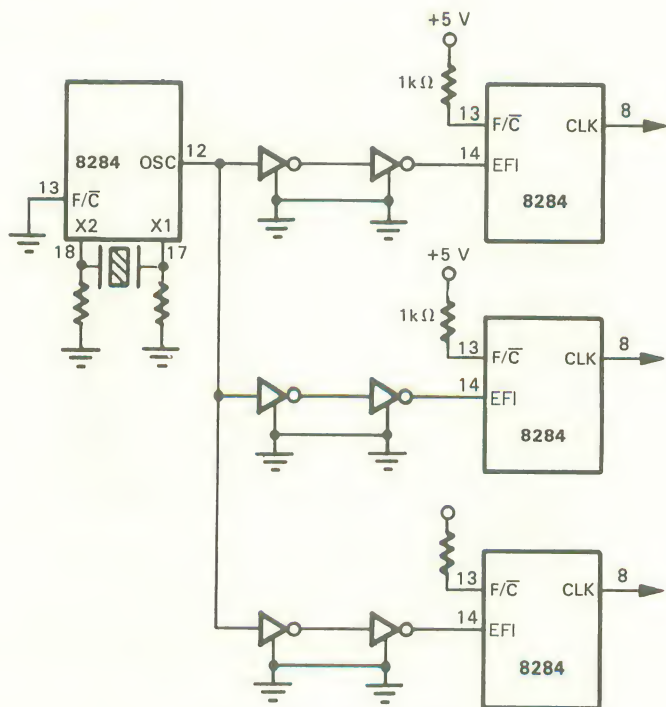


図8-10 マスタ周波数源の発生

各ラインは、74LS04などのバッファで駆動されるツイスト・ペアとし、そのグラウンドはソースとレシーバのグラウンドに接続しなければならない。クロックのスキューを最小にするためには、すべての8284に対するラインの長さは等しくなければならない。付加された8284に対するメインの周波数源を発生する簡単な方法を図8-10に示す。

図8-10において、1つの8284はクリスタルで必要な周波数を発生させるために用いられている。8284のオシレータ出力（OSC）はクリスタルの周波数に等しく、システムの他のすべての外部周波数入力を駆動するために用いられる。

オシレータの信号のコンプリメントがCPUクロック・ジェネレータ回路の駆動に用いられるように、OSCは反転している。したがって、2つの8284が同期すべき別々のCPUのクロック入力を駆動するならば、1つの8284のOSCでもう1つの8284のEFI入力を駆動することはできない。8284の範囲でEFI対CLKの遅延の変動は35nsから45nsに近く、しかし、すべての8284が同一のパッケージ・タイプで、同一の相对供給電圧で、同じ温度環境で動作するならば、変動は15nsと25nsの間にまで減少する。

8284には、前述のOSC、CPUを駆動するシステム・クロック（CLK）、そしてCPUのクロック周波数の $\frac{1}{2}$ で動作する周辺装置用クロック（PCLK）の3つの周波数出力がある。OSCはクリスタルで駆動されているだけで、F/Cの接続による選択の影響を受けな

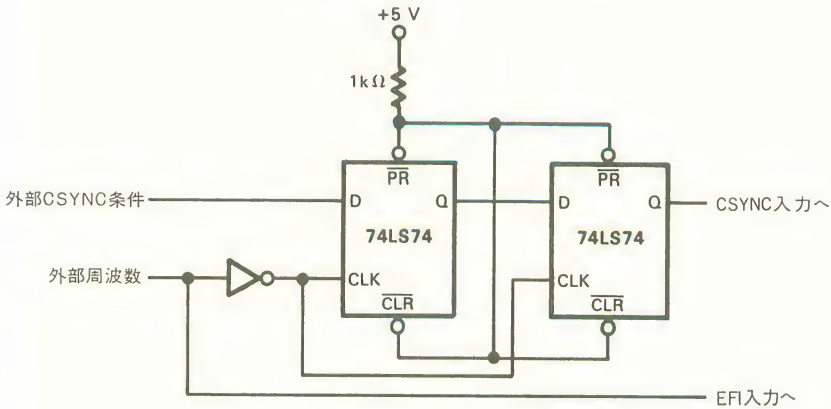
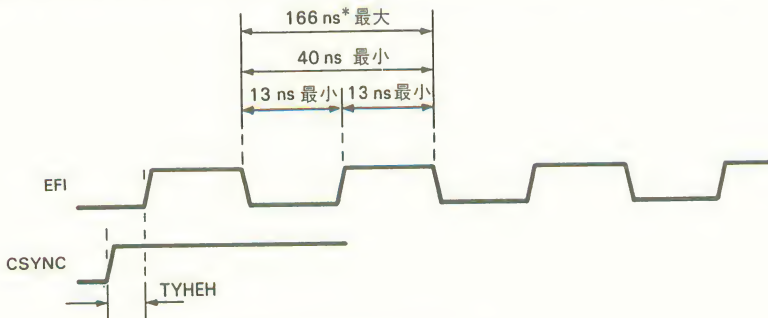


図8-11 CSYNCの同期

い、外部周波数入力を用いられてクリスタルが8284に接続されていなければ、OSCは不定となる。CLKは、選択された周波数源から3進カウンタの内部分周によって得られる。カウンタは、最大周波数のCPUに最適な33%デューティ・サイクルのクロックを発生する。PCLKは50%デューティ・サイクルで、CLKの周波数の $\frac{1}{2}$ で動作する。

3進カウンタで分周される8284の状態はシステム初期化（パワー・オン）では不定なので、CPUクロックの外部事象に対する同期が可能となるように、カウンタに対する外部同期信号（CSYNC）が備わっている。CSYNCがハイとなると、CLKとPCLKの出力は強制的にハイとなる。CSYNCがローに戻ると、同波数源の次の正のクロックでクロックの発生が開始する。CSYNCは、周波数源の最小2つの期間、アクティブでなければならない。CSYNCが周波数源と同期していなければ、同期のために図8-11の回路を用いる必要がある。

2つのラッチは、CSYNCを駆動するラッチにおいて準安定状態の可能性を最小にしている。図8-12に示すようにラッチは、周波数源に対する8284のセット・アップとホールド



\*最大値は、最低クロック周波数を保証するために指定される。

図8-12 CSYNCのタイミング

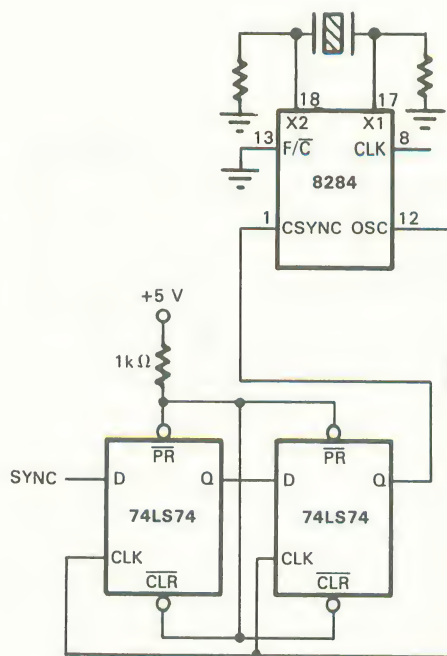


図8-13 OSCを用いたCSYNCの同期

の時間を保証するために、周波数源の反転をクロックとしている。

1つの8284が外部事象に同期の必要があり、外部周波数源が用いられていなければ、図8-13に示すように、8284のOSCがCSYNCを同期させるために用いられる。

OSCは内部オシレータの信号に関して反転しているので、前の例でのインバータは必要でない。複数の8284を同期させる必要があれば、図8-14に示すように、外部周波数ですべての8284を駆動し、1個のCSYNC同期回路ですべての8284のCSYNC入力を駆動しなければならない。

CSYNCがアクティブのとき、8086クロックのローの最小時間は十分でないので、リセットの期間だけあるいはCPUのクロックがハイの間は、イネーブルとしなければならない。CSYNCはまた、適当なCPUのリセットを保証するために、リセットの終了前に最小4クロックの期間、デイスエーブルでなければならない。

8284のCLK出力における高速の変化と高い駆動力(5mA)のために、リングングの除去にクロック・ラインと直列に100オームの抵抗を入れる必要がある。スキューが最小のCLKの複数のソースが必要ならば、CLKは、最小 $V_{CH}=3.9$ (8086の入力がハイの最小電圧)を保証するために、出力が100オームを通して5ボルトに接続された、高い駆動力の素子(74S241)によるバッファを用いることができる。8284は、CPUに対するREADYの同期をとり、1つのCPUに対するREADYを供給するだけなので、1つの8284は複数のCPUのCLKを発生するためには用いることができない。



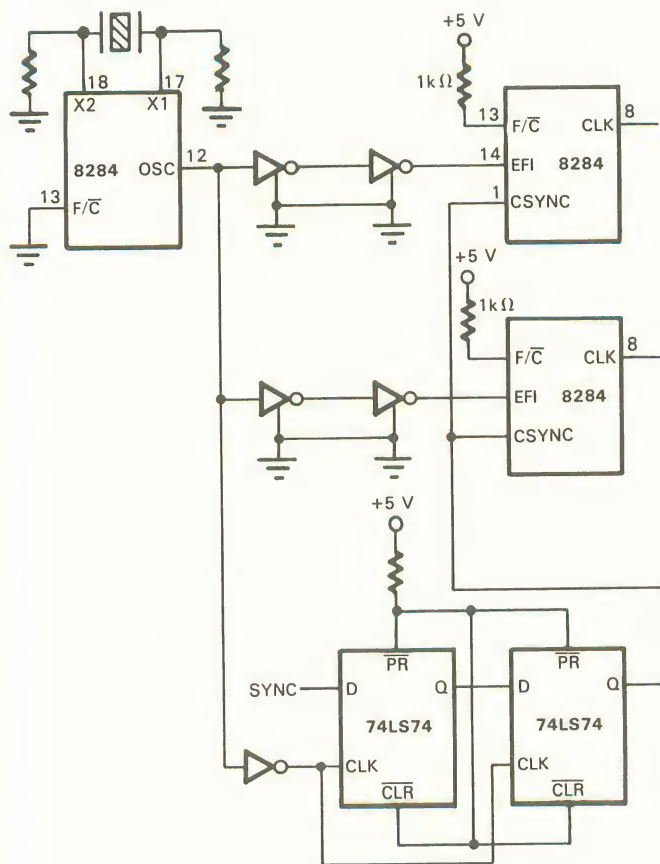


図8-14 複数の8284に対するCSYNCの分配

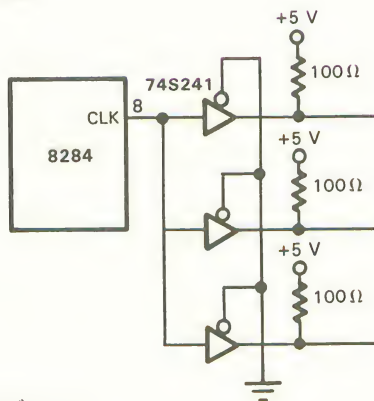


図8-15 高駆動力素子によるCLKのバッファ

### 8.3 リセット

8086は、 $50\mu\text{s}$ のリセット・パルスを必要とするパワー・オン後を除いて、最小パルス幅が4つのCPUクロック期間のアクティブ・ハイのリセットを必要とする。CPUは内部でリセットとクロックの同期をとるので、外部リセットがローになった後の1クロックの期間まで、リセットは内部的にアクティブである。ノンマスカブル・インタラプト (NMI), ミニマム・モードのホールド・リクエスト, あるいは内部リセットの間に生じるマキシマム・モードのRQパルスは、受け付けられない。内部リセット直後にアクティブなミニマム・モードのホールド・リクエスト, あるいはマキシマム・モードのRQパルスは、最初の命令フェッチの前に受け付けられる。

8086がリセットを認めると、CPUはバスを表8-4に示すような状態にする。

表8-4 リセット期間の8086バス信号

| 信 号                                           | 状 態                   |
|-----------------------------------------------|-----------------------|
| AD0-AD15                                      | トライステート               |
| A16-A19/S3-S6                                 | 不 定                   |
| BHE/S7                                        | 不 定                   |
| S2/(M/ $\overline{\text{IO}}$ )               | } “1”にした後に<br>トライステート |
| S1/(DT/R)                                     |                       |
| S0/( $\overline{\text{DEN}}$ )                |                       |
| $\overline{\text{LOCK}}/\overline{\text{WR}}$ |                       |
| $\overline{\text{RD}}$                        |                       |
| $\overline{\text{INTA}}$                      |                       |
| ALE                                           | 0                     |
| HLDA                                          | 0                     |
| $\overline{\text{RQ}}/\overline{\text{GT0}}$  | 1                     |
| $\overline{\text{RQ}}/\overline{\text{GT1}}$  | 1                     |
| QS0                                           | 0                     |
| QS1                                           | 0                     |

リセットが検出されると、CPUによって、多重化バス信号の接続はフロート状態となる。フロート状態にできる他の信号は、トライステートになる前のCLKの1つのロー状態の間、インアクティブ状態に駆動される。この様子を図8-16に示す。

ミニマム・モードにおいて、ALEとHLDAはインアクティブに駆動されるがフロートとはならない。マキシマム・モードでは、RQ/GTラインはインアクティブに保持され、キュー・ステータス出力(Q0とQ1)はアクティブでないことを示す。キュー・ステータスはキュー・リセットを表示しないので、キューをモニタするユーザ定義の外部回路もまた、

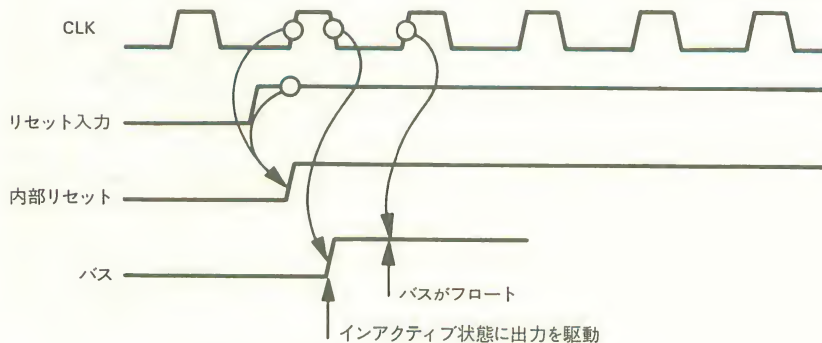


図8-16 リセットにおける8086バスの状態

システム・リセットによってリセットされなければならない。22キロオームのプルアップ抵抗をCPUのコマンドとバス・コントロールのラインに接続する必要がある。これにより、漏れ電流（あるいはバス・キャパシタンス）がシステムにおける素子の最小のハイ電圧以下に電圧レベルを下げてしまうシステムで、上記ラインのインアクティブ状態が保証される。

マキシマム・モード・システムでは、8288は $\overline{S0}$ — $\overline{S2}$ 入力に内部プルアップを含む。これにより、CPUがバスをフロート状態にしたときに、これらのラインのインアクティブ状態を維持する。リセットの間のステータス・ラインのハイ状態によって、8288はリセットのシーケンスをパッシブ状態として取り扱う。パッシブ状態に対する8288の出力状態を表8-5に示す。

表8-5 パッシブ状態の8288の出力

|                    |     |
|--------------------|-----|
| ALE                | 0   |
| DEN                | 0   |
| DT/ $\overline{R}$ | 1   |
| MCE/PDEN           | 0/1 |
| コマンド               | 1   |

バス・サイクルの間にリセットが起きると、ステータス・ラインはパッシブ状態に戻り、バス・サイクルは終わり、そしてコマンド・ラインはインアクティブになる。8288は、ステータス・ラインのパッシブ状態に基づいてコマンドの出力をフロートにしないことに注意。シングルCPUシステムにおけるリセットの間に、CPUをバスから切り離す必要があれば、図8-17に示すように、リセット信号はまた8288のAEN入力とアドレス・ラッチの出力イネーブルに接続しなければならない。

この方法は、8288からのDENのインアクティブ状態がデータ・バスのトランシーバをフロートにする間、コマンドとアドレスのバス・インターフェイスをフロートする。

マイクロプロセッサ共有バス結合を確立するために判定を用いる複数プロセッサ・シス

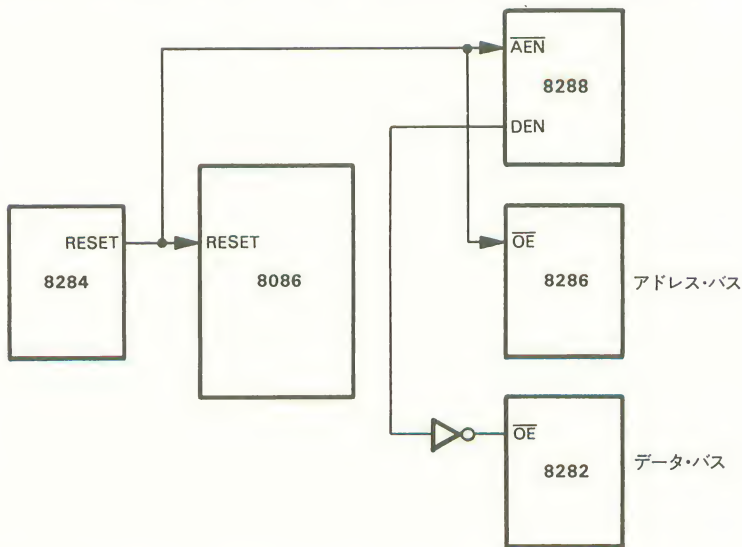


図8-17 マキシマム・モード8086バス・インターフェイスのリセット・ディスエーブル

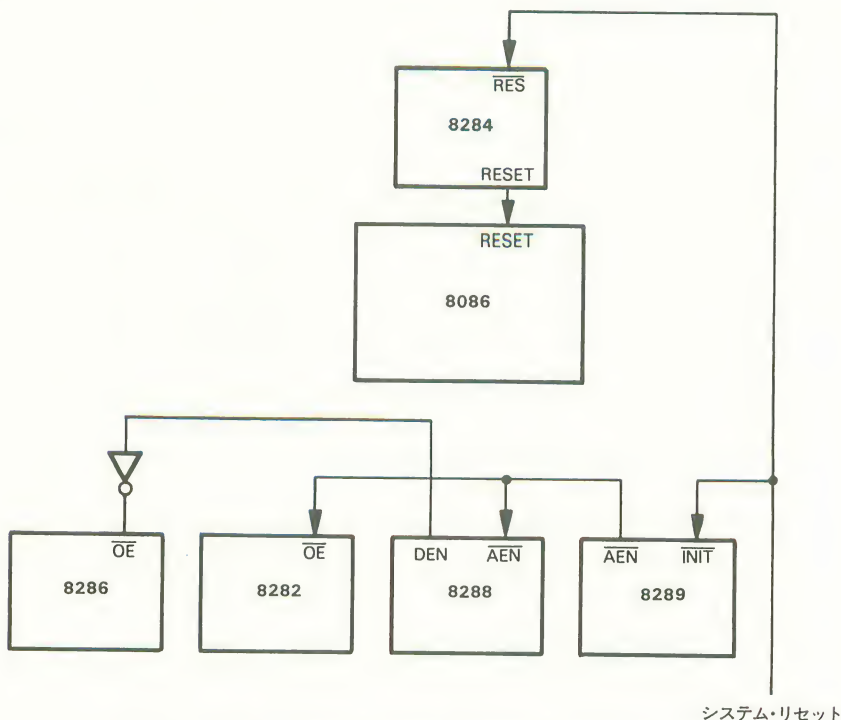


図8-18 マルチCPUシステムにおけるマキシマム・モード8086バス  
・インターフェイスのリセット・ディスエーブル

テムにおいて、図8-18に示すように、システム・リセットは、8284のリセット入力に加えて、8289バス・アービタのINIT入力に接続されなければならない。

アクティブ・ローのINIT入力により、すべての8289の出力はインアクティブ状態となる。インアクティブ状態の8289 AEN出力は、8288のコマンド出力をフロート状態にさせ、さらに、アドレス・ラッチはアドレス・バス・インターフェイスをフロートにする。1つ以上のマイクロプロセッサがマスタとして機能するマルチマイクロプロセッサ・システムでは、リセットはすべてのCPU、8289、8284に共通でなければならない。このリセットはすべてのCPUのリセット条件、あるいは3つの8289バス・クロック期間(TBLBL)と3つの8086クロック期間を満たす必要がある。これは8289のリセット条件を満たしている。

リセットの間は8288のコマンド出力はフロートとなり、コマンド・ラインは2.2キロオームの抵抗を通して  $V_{cc}$  に接続されていなければならない。

8086に対するリセット信号は8284から得られる。8284は、アクティブ・ローの外部リセットからリセットを生成するために、8284はシュミット・トリガ入力を持っている。8284データ・シートに記されているヒステリシスは、少なくとも0.25ボルトが8284リセット入力の0と1のスイッチング・ポイントを分離することを示している。ヒステリシスのない

入力は、近似的に同一電圧のスレッシュホールドでローからハイとハイからローにスイッチする。

入力は、明記されたローとハイの電圧 ( $V_{IL}$  と  $V_{IH}$ ) でスイッチすることが保証されているが、実際のスイッチング・ポイントはこの間のどこかにある。  $V_{ILmin}$  は0.8ボルトと指定されているので、ヒステリシスにより、入力が少なくとも1.05ボルトに達するまで、リセットがアクティブであることが保証される。リセット入力である2.6ボルトの  $V_{IH}$  以下に少なくとも0.25ボルト、入力が下がるまでリセットは認められない。

パワーアップからのリセットを保証するためには、リセット入力は、 $V_{ce}$  が4.5ボルトの最小供給電圧に達した後に、50マイクロ秒の間は1.05ボルト以下でなければならない。ヒステリシスにより、リセット入力は図8-19に示す簡単なRC回路で駆動できる。

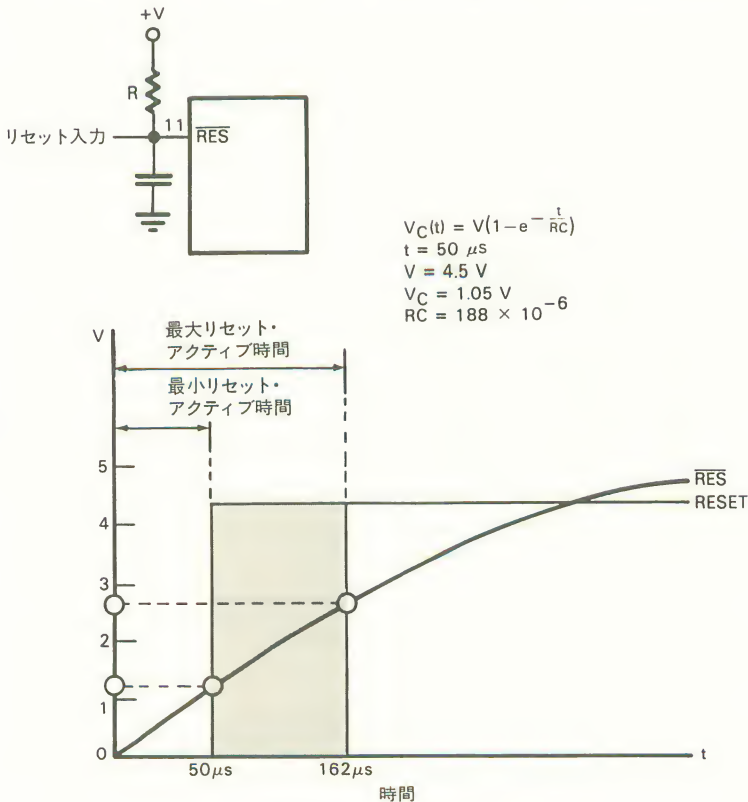


図8-19 8284のリセット回路

計算されたRC値は、電源供給が4.5ボルトに達する時間あるいはこの期間に蓄積される電荷を含んでいる。ヒステリシスがなければ、入力電圧が入力のスイッチング電圧を通過するのに従って、リセット出力は振動する。計算されたRC値は、1.05ボルトのレベルでスイッチする8284の50マイクロ秒のリセット期間と、2.6ボルトのレベルでスイッチする82



84の約162マイクロ秒のリセット期間に必要な最小値を与える。最小と最大のリセット時間の間により小さい許容差が必要ならば、簡単なRC回路よりも、図8-20に示されているリセット回路が用いられる。

図8-20に示す回路は、RC回路の逆指数関数的な充電速度ではなく、定電流源でキャパシタに対する線形充電速度を与える。この回路の最大リセット期間は124マイクロ秒である。

図8-21に示すように、CPUに対するリセット信号を発生するために、8284はリセット入力とCPUクロックの同期をとる。

出力はまたシステム全体の全般的なリセットとして用いられる。リセットは、8284のクロック回路には何の影響も与えない。

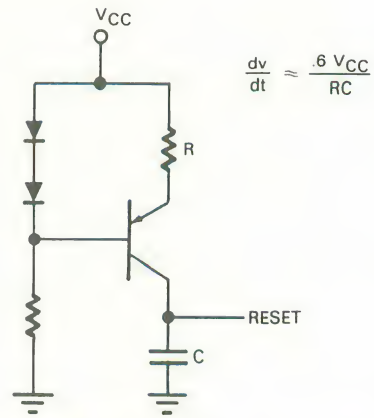


図8-20 定電流パワーオン・リセット回路

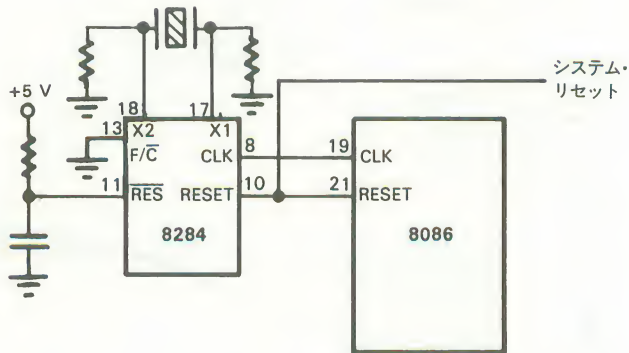


図8-21 8086のリセット

## 8.4 レディの実現とタイミング

最大のCPUバス帯域幅で情報を伝送できないメモリとI/O素子を適応させるために、8086はREADY信号を用いる。READYはまた、マルチマイクロプロセッサ・システムで8086のシステム・バスに対するアクセスをウエートさせるために用いられる。バス・サイクルにウエート状態を挿入するために、CPUに対するREADY信号は、T2の終わりまでにインアクティブ（ロー）でなければならない。ウエート状態の挿入を避けるためには、T3期間の正の遷移の前に指定されたセットアップ時間内に、READYはアクティブ（ハイ）でなければならない。システムの規模と特性に依存して、READYのロジックは次の2つ

の方法の1つをとる。

- (1) システムは通常、ノット・レディである。選択されたメモリまたはI/O素子でデータ伝送の用意ができると、ハイのREADY信号を入力する。
- (2) システムは通常、レディである。選択されたメモリまたはI/O素子がCPUの最大伝送速度でデータ伝送ができなければ、ローのREADY信号を入力しなければならない。

“正統派的”なREADYの実現は、システムを“通常はノット・レディ”の状態に保つ。選択された素子が読み込み/書き込み、またはインタラプト・アクノリッジのコマンドを受け取ると、このコマンドに応答するのに十分な時間があれば、8086にREADYのハイを入力する。これにより、8086はバス・サイクルを進めることができる。

この実現は、規模の大きいマルチマイクロプロセッサ、マルチバス・システム、あるいは伝播遅延、バス・アクセス遅延、さらに素子の特性が固有にシステムを遅くする場合の特徴となる。この方法を用いて、ウエート状態なしで動作できる素子は、最大のシステム性能に対して前述の制限以内にREADYのハイを返さなければならない。高速の素子が時間内に応答できなければ、バス・サイクルにウエートのクロック期間が挿入される。

代替りの方法は、“通常はレディ”のシステムを持つことである。すべての素子は、最大のCPUバス帯域幅で動作することを仮定している。必要条件を満たさない素子は、ウエート状態のクロック期間の挿入を確実にするために、T2の終わりまでにREADYのローを入力しなければならない。この実現は一般に、規模の小さいシングルCPUシステムに適用される。これにより、READY信号のコントロールに必要なロジックが減少する。ウエート状態を要する素子がT2の終わりまでにREADYをデイスエーブルにできなければ、バス・サイクルを早まって終結させることになるので、この方法を用いるときは、システムのタイミングを十分に解析しなければならない。

8086システムでは、システムの性能の最適化のために、シングル・システムにおいて前述の2つのREADYの方法を設計者が結合できることを10章に示す。

8086は、システムの実現に依存して、READYに対する2つの異なるタイミング条件を持つ。“通常はノット・レディ”のシステムでは、ウエート状態を避けるために、T3の間の

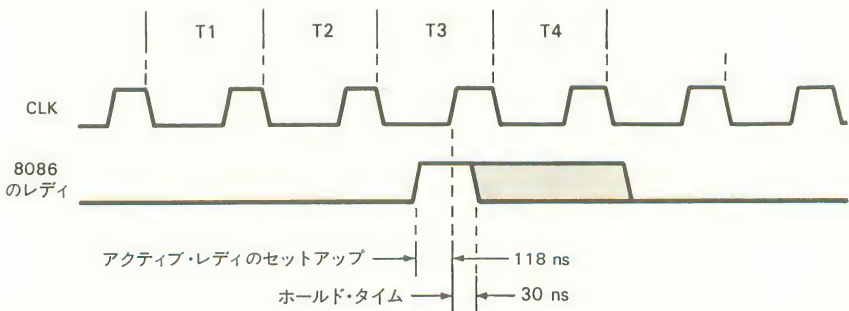


図8-22 ウエート状態を避ける通常ノット・レディのシステム

正のクロック遷移の118ns (TRYHCH) 以内に READY はハイでなければならない。この様子を図8-22に示す。

“通常はレディ”のシステムは、図8-23に示すように、T2の終わり(T3の初め)の後の8ns (TRYLCL) 以内に READY のローを入力して、ウェート状態を挿入しなければならない。

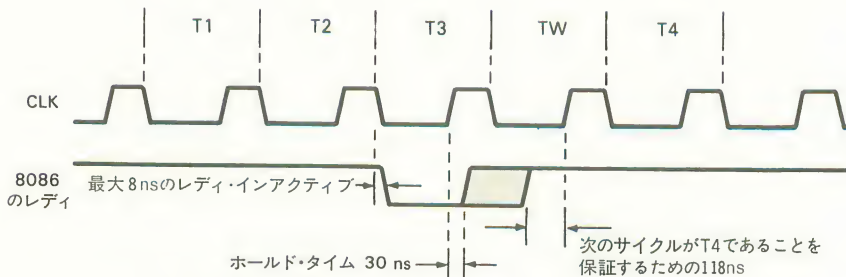


図8-23 ウェート状態を挿入する通常レディのシステム

8086の正しい動作を保証するために、T3のクロックがローの期間は、READY 入力をハイからローに変化させてはならない。両方の場合において、READY は、T3の正のクロックの遷移から30ns (TCHRYX) のホールド・タイムを満たさなければならない。

前述のセットアップとホールド・タイムを満たす安定な READY 信号を得るために、8284は2つの分離したシステム・レディ入力 (RDY1, RDY2) と1つの同期レディ出力 (READY) を備えている。RDY 入力は別々のアクセス・イネーブル ( $\overline{\text{AEN1}}$ ,  $\overline{\text{AEN2}}$ ) とのゲートを持つ。これにより、図8-24に示すように、2つの READY 信号から1つを選択することができる。ゲートを持つ RDY 信号は論理的に8284によって OR がとられ、CPU に対する READY を発生するために、各 CLK サイクルの開始でサンプリングされる。このタイミングを図8-25に示す。

サンプルされた READY 信号は、“ノット・レディ”と“レディ”についての CPU のタイ

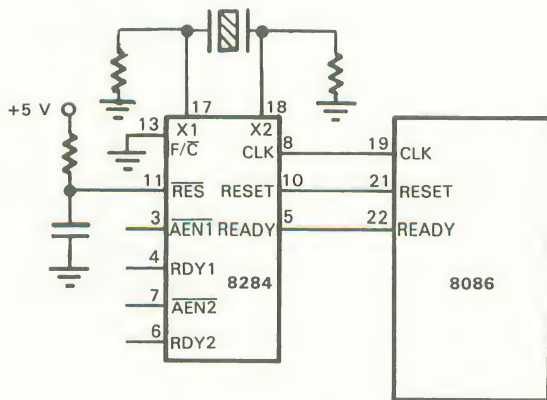
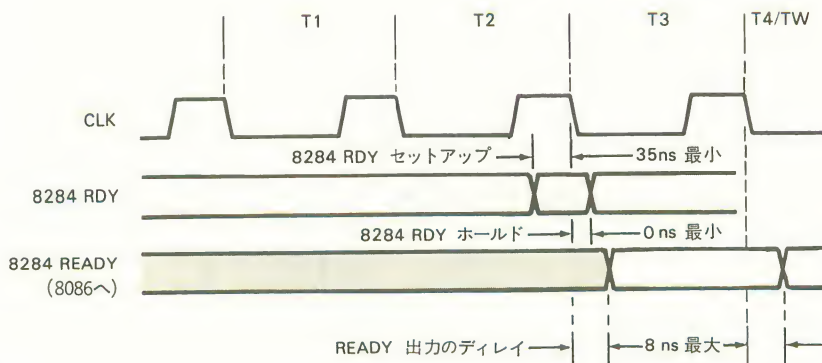


図8-24 8284と8086のレディの接続



注) 8284のデータ・シートには、レディ出力のディレイ (TRYLCL)は、T2の終わりの前-8nsと記されている。これは図に示されているタイミングを意味する。

図8-25 8284と8086のレディのタイミング

ミング条件を満たすために、CLKの後の8ns (TRYLCL) 以内は有効である。READYは次のCLKまで変化できないので、ホールド・タイムの条件も満たされる。8284に対するシステムのレディ入力 (RDY1, RDY2) はT3より35ns (TRIVCL) 前で有効で、 $\overline{\text{AEN}}$ はT3より60ns前で有効でなければならない。1つのRDY入力のみを用いたシステムでは、図8-26に示すように、関連のある $\overline{\text{AEN}}$ はグランドに接続され、もう1つの $\overline{\text{AEN}}$ は1キロオームを通して5ボルトに接続される。

システムがロー・アクティブのレディ信号を生成すると、8284の $\overline{\text{AEN}}$ 入力によって必要とされる付加的なセットアップ・タイムが満たされれば、それは8284の $\overline{\text{AEN}}$ 入力に接続することができる。この場合、図8-27に示すように、関連のあるRDY入力はハイに接続される。

CPUの最大周波数以下で動作するメモリと周辺素子の大多数は一般に、1つ以上のウェート状態を必要としない。図8-28の回路は1つのウェート状態を発生する。

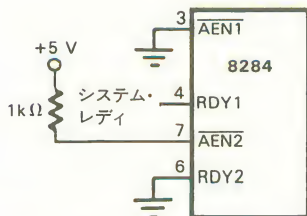


図8-26 1つのRDY入力を用いた8284

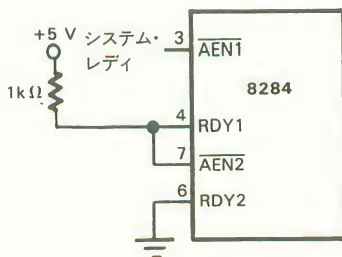


図8-27 アクセス・イネーブル駆動のシステム・レディを有する8284

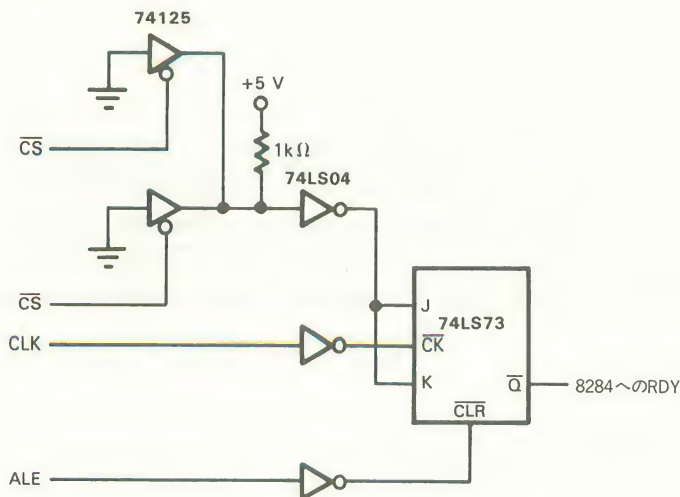


図8-28 シングル・ウェート状態の発生回路

図8-28のシステム・レディ・ラインは、1つのウェート状態を要する素子が選択されると常にローに駆動される。フリップフロップはALEでクリアされ、8284に対するRDYをイネーブとする。ウェート状態が必要でなければ、フリップフロップは変化しない。システム・レディがローに駆動されると、フリップフロップはT2のローからハイへのクロック変化で反転して、1つのウェート状態を強制する。CLKの次のローからハイへの変化でフリップフロップは再び反転し、レディを示してバス・サイクルの終了を可能にする。フリップフロップの状態のこれ以上の変化はバス・サイクルに影響しない。図8-29に示すように、回路はシステム・レディに対するチップ・セレクトに約100nsの余裕がある。

システムが“通常はノット・レディ”ならば、物理的なメモリの最後の6バイトにプログラムは実行可能コードを割り当てられない。8086は命令を前もってフェッチするので、物

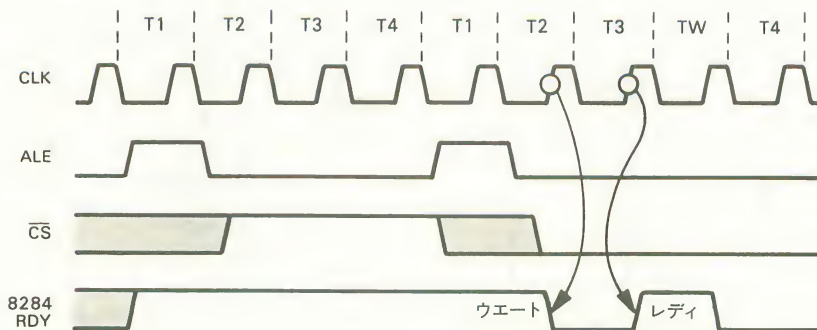


図8-29 シングル・ウェート状態の発生回路のタイミング



理的なメモリの終わりのコードを実行するときは、CPUは存在しないメモリをアクセスしようとする。存在しないメモリに対するアクセスがREADYをイネーブルにできなければ、システムは不定のウェイト状態に陥る。

## 8.5 インタラプト構造

8086のインタラプト構造は、図8-30に示されているように、メモリ位置  $0_{16}$  から  $003FF_{16}$  まではストアされるインタラプト・ベクタのテーブルに基づいている。各ベクタは4 バイ

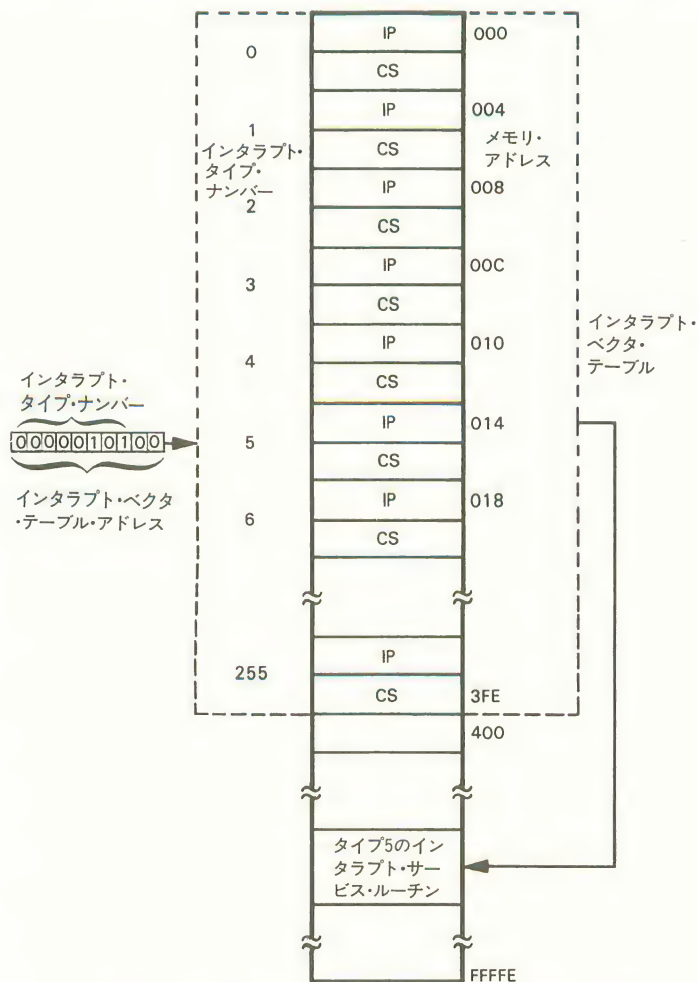


図8-30 インタラプト・ベクタ・テーブルからのインタラプト・サービス・ルーチン・アドレスの獲得

トで、最初の2バイトは新しいプログラム・カウンタ・アドレスを、次の2バイトは新しいコード・セグメント・レジスタ・アドレスを保持している。

この2つのアドレスは結合されて、インタラプト・サービス・ルーチンの20ビットの実行アドレスを形成する。この20ビットのアドレスは、通常の8086のセグメント・プログラム・メモリ・アドレッシングを用いて計算される。インタラプト・ベクタ・テーブルは、8086の1メガバイトのアドレス空間の任意の場所に位置するインタラプト・サービス・ルーチンの開始アドレスを指定する、256までのインタラプト・ベクタを含む。

特定の構成で256以下のインタラプトを用いるならば、用いられているインタラプト・ベクタのメモリを割り当てただけでよい。しかしシステムのデバッグを行なうときは、誤った割り込みを検出する手段として、すべての未定義の割り込みをトラップ・ルーチンに割り当てなければならない。

各インタラプト・ベクタは、関連のあるインタラプト・ナンバーを持つ。インタラプト・ナンバーは、インタラプト・ベクタ・テーブル内のインタラプト・ベクタを識別する。インタラプト・ナンバーに4を乗じることによって、インタラプト・ベクタ・テーブル内のインタラプト・ベクタのエントリの最初のバイトに対する絶対アドレスが得られる。たとえば、インタラプト・ナンバー5は、インタラプト・ベクタ・テーブルの6番目のエントリを示し、このベクタの最初のバイトはアドレスが $20_{16}$ (= $14_{16}$ )である。この様子を図8-30に示す。

このようにして、8086のインタラプト構造により、各インタラプト・サービス・ルーチンに対する開始メモリ・アドレスが指定できる。

8086で特殊な機能によって要求される定義済のインタラプト、ユーザ定義のハードウェア・インタラプト、ソフトウェア・インタラプトの3つの型のインタラプトが8086にはある。

定義済インタラプトは、ハードウェアやソフトウェアによって要求することができる。次に定義済インタラプトについて詳細に検討する。

### 8.5.1 定義済インタラプト

“定義済”インタラプトは、割り当てられたインタラプト・ナンバーと自動的にベクタを決定するロジックを持つことからそう呼ばれる。したがって、定義済インタラプトが受け付けられると、8086のロジックは自動的に、インタラプトの割り付けられたベクタ・テーブル・エントリのベクタ決定を行なう。ただし、プログラム・カウンタとコード・セグメントのアドレスでベクタ・テーブル・エントリを初期設定して、各インタラプトにインタラプト・サービス・ルーチンを用意しなければならない。

外部ロジックによって要求される定義済ハードウェア・インタラプトと、命令実行の結果として要求される定義済ソフトウェア・インタラプトが存在する。

インタラプト・ナンバー0から31までは、定義済インタラプトに割り当てられている。定義済インタラプトを用いなければ、そのインタラプト・ナンバーを何か他のインタラプトに用いることができる。しかし、このようなシステムは、将来における8086のハードウェアとソフトウェアの製品との互換性がなくなるので、推奨されない。

次に定義済インタラプトを1つずつ述べる。

### (1) インタラプト 0 — 0による除算

このインタラプトは、除算命令の実行に続いて、商が除算命令で許される最大値を越えると自動的に要求される。このインタラプトはマスクすることができない。これは標準の除算命令実行ロジックの一部として要求される。0による除算のインタラプト・サービス・ルーチンによってインタラプトが再びイネーブルにされなければ、“ワースト・ケース”の除算命令時間の計算に、このサービス・ルーチンの実行時間が含まれる。これは除算命令に対して最も長い実行時間となる。

### (2) インタラプト 1 — シングル・ステップ

このインタラプトは、プログラム・ステータス・ワードでTF(トラップ・フラグ)がセットされて1命令後に発生する。このインタラプトは、プログラムを一度に1命令実行させるために用いられる。プログラムの各命令が実行された後で、インタラプトが要求される。インタラプトの要求に続いて、次に実行されるプログラムの命令の結果に、インタラプト・サービス・ルーチンによって種々の診断機能が行なわれ、そして次のシングル・ステップのインタラプト要求が発生する。

シングル・ステップの開始には、プログラム・ステータス・ワードの内容をスタックにプッシュして、スタックのトップにセーブされたプログラム・ステータス・ワード内のトラップ・フラグをセットして、プログラム・ステータス・ワードにスタックからポップする。シングル・ステップのインタラプトは、次の命令の実行に続いて要求される。

シングル・ステップのインタラプト要求が受け付けられると、シングル・ステップのインタラプト・サービス・ルーチン自身がシングル・ステップのインタラプト要求によるインタラプトを受けないように、プログラム・ステータス・ワード中のTFフラグはリセットされる。スタックにセーブされているフラグのTFはセットされたままである。

シングル・ステップのインタラプト・サービス・ルーチンからの復帰にはIRET命令を用いなければならない。この復帰はフラグ(TFを含む)を復元して、別のTFインタラプトが次の命令の終了で発生することを可能にする。

### (3) インタラプト 2 — NMI(ノンマスクابل・インタラプト)

これは最も高い優先度のハードウェア・インタラプトである。その名前が意味するように、マスクすることはできない。NMIインタラプト要求の入力は、NMI入力のローからハイへの遷移によるエッジ・トリガであり、内部的にCPUのクロック信号CLKのローからハイへの遷移と同期している。したがってNMIは、認識を保証するために少なくとも2クロック期間、ハイでなければならない。NMI入力のローからハイへの遷移でインタラプト要求を生成できるので、擬似的な遷移は抑圧する必要がある。

NMIが通常ハイならば、認識を保証するためにアクティブなローからハイへの遷移を行なう前に、2つのCPUクロック期間はローでなければならない。この入力は一般に、たとえばパワー低下やシステムのウォッチドッグ・タイマの時間超過に続く、非常時のインタラプト要求のために保留されている。

#### (4) インタラプト 3 — 1 バイト・インタラプト

これはソフトウェア・インタラプトである。これは、1 バイトのオブジェクト・コードを占める特殊なインタラプト要求命令の実行によって発生する。これは、ソフトウェア・デバッグ・プログラムでブレイクポイントを設定するために用いられる。最小の8086命令オブジェクト・コードは1 バイトなので、1 バイト・インタラプトはブレイクポイントを設定する手段としてどの8086命令とも置き換えられる。

この1 バイト・インタラプトはマスクできない。

#### (5) インタラプト 4 — インタラプト・オン・オーバーフロー

このインタラプト要求は、プログラム・ステータス・ワードのオーバーフロー・フラグ (OF) がセットされて、INTO 命令が実行されると発生する。INTO 命令により、8086 のオーバーフロー・エラー・サービス・ルーチンへのトラップが可能になる。インタラプト・オン・オーバーフローはマスクできない。

### 8.5.2 ユーザ定義ソフトウェア・インタラプト

2 バイト・インタラプトの INT nn 命令の実行によって、ソフトウェア・インタラプトを起こさせることができる。最初のオブジェクト・コード・バイトはINTのオペコードで、次のオブジェクト・コード・バイト (nn) は実行されるべきインタラプトのナンバーを含む。INT 命令はマスクできない。

この命令は、ダイナミックにリロケート可能なプログラムを呼び出すためによく用いられ、メモリでの呼び出されたプログラムの位置は呼び出し元プログラムに知られていない。ただし、呼び出されるプログラムがメモリにロードされると、その実行アドレスは、インタラプト・ベクタにロードされる。呼び出されたプログラムは、インタラプト・リターン (IRET) 命令で復帰しなければならない。

### 8.5.3 ユーザ定義ハードウェア・インタラプト

マスク可能なハードウェア・インタラプトは8086のINTRピンによって要求され、このインタラプトはプログラム・ステータス・ワードのIFビット (インタラプト・フラグ) によってマスクできる。各命令実行の最後のクロック期間に、INTRピンの状態がサンプルされる。この規則には次の2つの例外がある。

1. 命令がセグメント・レジスタへのMOVあるいはセグメント・レジスタへのPOPであるとき。
2. それが先行する命令の一部として取り扱われる命令のプレフィックスの実行の間。

この2つの例外は、“一般的な場合”のインタラプト・アクノリッジ実行シーケンスの記述に続いて述べる。

### 8.5.4 インタラプト・アクノリッジ・シーケンス

“一般的な場合”としてユーザ定義のハードウェア・インタラプトを取り上げて、インタラプト・アクノリッジ・シーケンスについて述べる。



サンプル時にINTR信号がハイでプログラム・ステータス・ワードのIFビットが1ならば、ユーザ定義のインタラプトは要求される。このインタラプトはイネーブルなので、8086はインタラプト・アクノリッジ・シーケンスを実行する。インタラプトが受け付けられるのを保証するために、ミニマム・システムでは $\overline{\text{INTA}}$ によって、マキシマム・システムでは $\overline{\text{S0}}$ ,  $\overline{\text{S1}}$ ,  $\overline{\text{S2}}$ によって、8086がインタラプト・アクノリッジを返すまで、INTR入力はハイに保持されなければならない。

現在の命令が実行を終了するときBIUが命令をフェッチしているならば生じるような、BIUがインタラプト条件の検出されるバス・サイクルならば、INTRのインタラプト・リクエストは、バス・サイクルのT4より前の2クロック期間は無効でなければならない。そうでなければ、インタラプト・アクノリッジが発行される前に(1つは保留でも)別のバス・サイクルが実行される。

ロックされた命令の実行中にインタラプトとホールドが要求されたならば発生するように、ホールド・リクエストが保留されていれば、ホールドが最初に処理され、ホールドの処理が終わってからインタラプトが受け付けられる。

INTRピンに生じるユーザ定義ハードウェア・インタラプトの要求だけが特定のハードウェア・アクノリッジを受け取る。このアクノリッジは、図8-31に示されているように、2つのアイドル・クロック期間によって分離された、2つのインタラプト・アクノリッジ・バス・サイクルの形をとる。ソフトウェア・インタラプトとノンマスクابل・インタラプトは、図8-31に示されているアクノリッジ・シーケンスを受け取らない。

図8-31に示されているように、完全なインタラプト・アクノリッジ・シーケンスは、2つのアイドル・クロック期間で分離された、2つの $\overline{\text{INTA}}$ バス・サイクルから成る。2つのバス・サイクルの間、 $\overline{\text{INTA}}$ はインタラプトのアクノリッジのために(ミニマム・モードでは)ローが出力される。アドレス/データ・バス( $\overline{\text{BHE}}$ を含む)と関連のあるステータス(S3-S7)は2つのバス・サイクルの間、フロート状態となる。しかし、ハイのAL

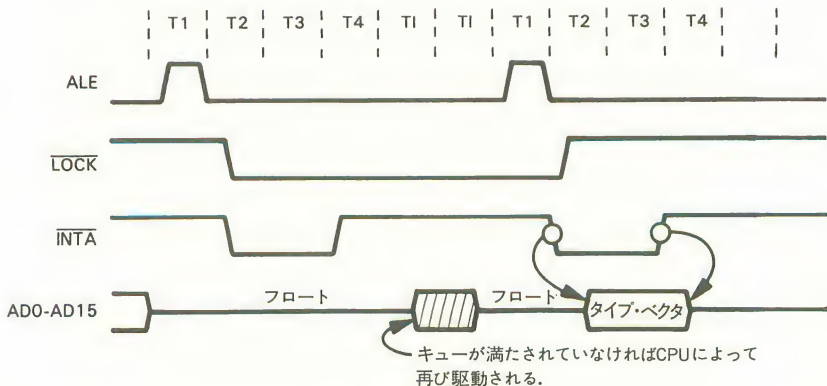


図8-31 ミニマム・モードにおけるインタラプト・アクノリッジ・シーケンス



Eパルスが出力されるので、アドレス・ラッチには不確定の情報がロードされる。したがって素子は、出力を駆動する前に制限として、常に $\text{READ}(\overline{\text{RD}})$ のローを用いるべきである。

$\overline{\text{INTA}}$ バス・サイクルの間、 $\text{DT}/\overline{\text{R}}$ と $\overline{\text{DEN}}$ はアクティブである。これにより、インタラプトを要求する素子からの1バイトのインタラプト・ナンバーを8086が受け取ることができる。

最初のバス・サイクルはインタラプト・アクノリッジが進行中であることを知らせる。これにより、次の $\overline{\text{INTA}}$ バス・サイクルの間に伝送するためのインタラプト・ナンバーを用意する時間を、インタラプトを発行した素子に可能とする。インタラプト・ナンバーは、第2の $\overline{\text{INTA}}$ バス・サイクルの間に16ビット・データ・バスの下位で8086に伝送されなければならない。したがって、インタラプト・ベクタを与える素子は、16ビット・データ・バスの下位に接続しなければならない。

$\overline{\text{INTA}}$ バス・サイクルのタイミング（アドレスのタイミングを除く）は、リード・バス・サイクルのタイミングと同様である。

8086のインタラプト・アクノリッジ・シーケンスは、インタラプト・アクノリッジ・シーケンスの間にCPUによって読み出される命令のない8080や8085とは異なっていることに注意。8080と8085は、アクノリッジ・シーケンスの一部としてCPUに対してインタラプトを発行した素子によって出されるリスタートあるいはコールの命令を必要とする。

ミニマム・モード・システムにおいて、インタラプト・アクノリッジ・バス・サイクルの間、 $\text{M}/\overline{\text{IO}}$ 信号はローである。

8086は、BIUが2つの $\overline{\text{INTA}}$ サイクルの間に発生するホールド・リクエストを受け付けるのを防止している。

マキシマム・モード・システムでは、ステータス・ライン $\overline{\text{S0}}-\overline{\text{S2}}$ により、8288バス・コントローラはインタラプト・アクノリッジ・バス・サイクルの間、 $\overline{\text{INTA}}$ のローを出力する。どちらの $\overline{\text{RQ}}/\overline{\text{GT}}$ 入力におけるホールド・リクエストも8086が受け付けるのを防止し、マルチマスタ・システムにおいて2つのインタラプト・アクノリッジ・バス・サイクルの間にバス判定ロジックがバスを放棄するのを防止するために、最初のインタラプト・アクノリッジ・バス・サイクルのT2から、次のインタラプト・アクノリッジ・バス・サイクルのT2まで、8086の $\overline{\text{LOCK}}$ 出力はアクティブとなる。

8086は、セグメント・レジスタに対するMOVあるいはセグメント・レジスタへのポップの次にはINTRをサンプルしない。これにより、2つのロードをインタラプトが分離する可能性なしに、スタック・ポインタSSとSPのレジスタへの32ビット・ポインタのロードを可能にする。

次はインタラプトの起きない命令列の例である。

```
MOV    SS, NEW$STACK$SEGMENT
MOV    SP, NEW$STACK$POINTER
```

プレフィックスはそれが先行する命令の一部として取り扱われるので、プレフィックス命令の実行後に8086はINTRをサンプルしない。この規則の1つの例外は、ストリング・プリミティブにリピート(REP)プレフィックスが先行するときに起きる。繰返しのスト

リング操作は、各々の繰返しのスロリング・プリミティブの実行が終了するごとにINTRをサンプリする。これには、LOCK フレフィックスを持つ繰返しのスロリング操作が含まれる。複数のフレフィックスが繰返しのスロリング操作に先行し、命令がインタラフトで中断されると、スロリング・プリミティブ直前のフレフィックスだけが、インタラフト・ルーチンからの復帰に続いて回復させられる。プログラム実行の正しい再開を可能とするためには、次のプログラミング技法を用いなければならない。

|                     |      |                    |
|---------------------|------|--------------------|
| LOCKEDSBLOCK\$MOVE: | LOCK |                    |
|                     | REP  |                    |
|                     | MOVS | DEST,CS: SOURCE    |
|                     | AND  | CX,CX              |
|                     | JNZ  | LOCKEDSBLOCK\$MOVE |

MOVS 命令のために生成されるオブジェクト・コード・バイトは、(下降順に) LOCK フレフィックス、REP フレフィックス、セグメント変更フレフィックス、そして MOVS である。インタラフトから復帰すると、セグメント変更フレフィックスは、正しいメモリ位置の間で1つの付加的な伝送が起きることを保証するために回復させられる。移動に続く命令は、移動が完了したかを判断するために繰返しのカウント値を調べる。移動が完了していなければ、ブロック移動命令への復帰が起きる。

8086は、ハードウェア・インタラフトについてはバスから、ソフトウェア・インタラフトについては命令の流れから、インタラフト・ナンバーを読み込む。インタラフト・ナンバーは、インタラフト・ベクタ・テーブル中の対応するインタラフト・ベクタのアドレスを生成するために、4が乗じられる。インタラフト・ベクタの4バイトは次のものである。

- ・プログラム・カウンタの下位バイト
- ・プログラム・カウンタの上位バイト
- ・コード・セグメント・レジスタの下位バイト
- ・コード・セグメント・レジスタの上位バイト

次に、8086はプログラム・ステータス・ワードの内容をスタックにフッシュし、トラップとインタラフトのフラグをリセットして、それから現在のコード・セグメント・レジスタとプログラム・カウンタの内容をスタックにフッシュする。新しいコード・セグメント・レジスタとプログラム・カウンタの内容がインタラフト・ベクタ・テーブルからロードされる。すなわち、このためにリード・バス・サイクルが実行される。

インタラフト・アクノリッジ・シーケンスの間のインタラフト・ベクタ・テーブル参照では、セグメント・レジスタは用いられない。20ビットのアドレスを形成するために、ベクタのディスプレイメントが0に加えられ、セグメント・レジスタが選択されていないことを示すために、S4は1でS3は0となる。

以下は、ユーザ定義のマスク可能なインタラフトが受け付けられたときに実行される実際のバス・シーケンスである。

1. 2つのアイドル・クロック期間で分離された、2つのインタラフト・アクノリッジ・バス・サイクルが実行される。図8-31に示しているように、受け付けられた素子は、

- 第2のインタラプト・アクノリッジ・バス・サイクルの間に1バイトのデータとしてインタラプト・ナンバーを返す。このデータ・バイトは、2ビット左へシフトされて、インタラプト・ベクタの開始アドレスになる。
2. リード・バス・サイクルが実行され、この間に新しいCSレジスタの内容がインタラプト・ベクタの最初の2バイトから読み出される。
  3. リード・バス・サイクルが実行され、この間に新しいプログラム・カウンタの内容がインタラプト・ベクタの第3と第4のバイトから読み出される。
  4. ライト・バス・サイクルが実行され、この間にプログラム・ステータス・ワードの内容がスタックにフッシュされる。
  5. プログラム・ステータス・ワードのインタラプト(IF)とトラップ(TF)のフラグが0にリセットされる。これにより、マスク可能なものあるいはシングル・ステップのインタラプトがディセーブルとなる。
  6. ライト・バス・サイクルが実行され、この間にCSレジスタの内容がスタックにフッシュされる。
  7. ライト・バス・サイクルが実行され、この間にプログラム・カウンタの内容がスタックにフッシュされる。

次にプログラムの実行は、そのアドレスがインタラプト・ベクタからフェッチされている、インタラプト・サービス・ルーチンに分岐する。

ノンマスクابل・インタラプト、ソフトウェア・インタラプト、あるいはシングル・ステップ・インタラプトが受け付けられたときは、前記のステップ2から7が実行され、インタラプト・ナンバーは既知なのでステップ1は必要でない。

ユーザ定義のマスク可能なインタラプトが要求されたときの命令の終わりと、インタラプト・サービス・ルーチン実行の始まりとは、62クロック期間で分離されている。

ソフトウェア生成のインタラプトに対しては、インタラプト・アクノリッジ・バス・サイクルが実行されないことを除いて、同じシーケンスのバス・サイクルが実行される。結果として、インタラプト・サービス・ルーチンの実行に対するディレイは、INT nn とシングル・ステップで51クロック期間、INT3で52クロック期間、そして INTO で53クロック期間となる。

バス・サイクルにウエート状態が挿入されれば、前記のインタラプト・アクノリッジ・クロック期間の数は当然、それに応じて増加する。

次に、複数インタラプトとインタラプトの優先権について検討する。

INTRによる外部インタラプト要求のみがディセーブルとすることができる。結果的に、このインタラプトは最も低い優先権を持つ。他のインタラプトのアクノリッジ・シーケンスは、プログラム・ステータス・ワードのIFフラグをリセットする。したがって、INTRによって要求されたインタラプトは、他のインタラプトのサービス・ルーチンが完了するか、あるいはインタラプトが再びイネーブルとなる(IFフラグがセットされる)まで、受け付けることができない。

シングル・ステップを用いてデバッグを行なっているプログラムは、シングル・ステッ





先権が最も低くなる。

シングルステップは、優先権がNMIより高いかあるいは低いので、シングル・ステップ・インタラプト・サービス・ルーチンはその実行がNMIインタラプトに続いているかどうかを調べる必要がある。もしNMIインタラプトに続いていて、直ちにNMIの処理が必要ならば、シングル・ステップ・インタラプト・サービス・ルーチンは自分自身をディスエーブルにするロジックを含まなければならない。このプログラム・ロジックは、スタックのトップのリターン・アドレスを調べ、NMIインタラプト・サービス・ルーチンのアドレスを検出すると、NMIルーチンの実行を可能とするために復帰だけを行なう必要がある。NMIルーチンはシングル・ステップ状態のプログラムに復帰し、シングル・ステップは復帰の間にフラグが再現されて自動的に再びイネーブルとなる。実際の影響は、NMIが検出されると、シングル・ステップ状態のプログラムの1命令に対してシングル・ステップが無視されることである。シングル・ステップはインタラプト・アクノリッジの過程でディスエーブルなので、NMIインタラプト・サービス・ルーチンはその実行の間、シングル・ステップをディスエーブルとするために、プログラム・ステータス・ワードのTFフラグをリセットしておくことだけが必要である。

#### 8.5.5 システムのインタラプト構成

8259A プライオリティ・インタラプト・コントローラは、INTRを通して要求される複数の外部のユーザ定義インタラプトを取り扱うことができる。この素子は、8080A / 8085あるいは8086システムで動作する。8259Aは縦続接続が可能で、マスタ/スレーブ構成で、1つのシステムで64のインタラプトまで取り扱うことができる。

図8-32と図8-33に、ミニマム・モードとマキシマム・モードの8086システムにおける8259Aを示す。

図8-32aに示すミニマム・モード構成は、8086の多重化バスに接続されている8259Aを示す。図8-32bに示す構成は、分離バス・システムに接続されている8259Aを示す。これらの相互接続はまた、マキシマム・モード・システムにも適用できる。マキシマム・モード・システムに取り上げた構成は、付加のスレーブ8259Aがバッファを用いたシステム・バス上にあり、8086多重化バス上のマスタ8259Aを示している。この構成は、マキシマム・モード・システム・インターフェイスのいくつかの独特な特徴を表わしている。マスタ8259Aがスレーブ8259Aと正規の割り込みの素子の混合したインタラプトを受けると、スレーブは接続されている素子のインタラプト・ナンバーを供給する必要がある。一方マスタはそのインタラプト入力に直接所属しているインタラプト・ナンバーを供給しなければならない。

マスタ8259Aは、インタラプトが要求のある素子から直接かあるいはスレーブ8259Aから受けているものかを判断することができる。マスタ8259Aは、この情報をデータ・バス・トランシーバ(DENとENのNAND機能によって)をイネーブルあるいはディスエーブルとするために用いる。マスタ8259Aがインタラプト・ナンバーを供給しなければならないならば、データ・バス・トランシーバをディスエーブルにする。スレーブ8259Aがタイ



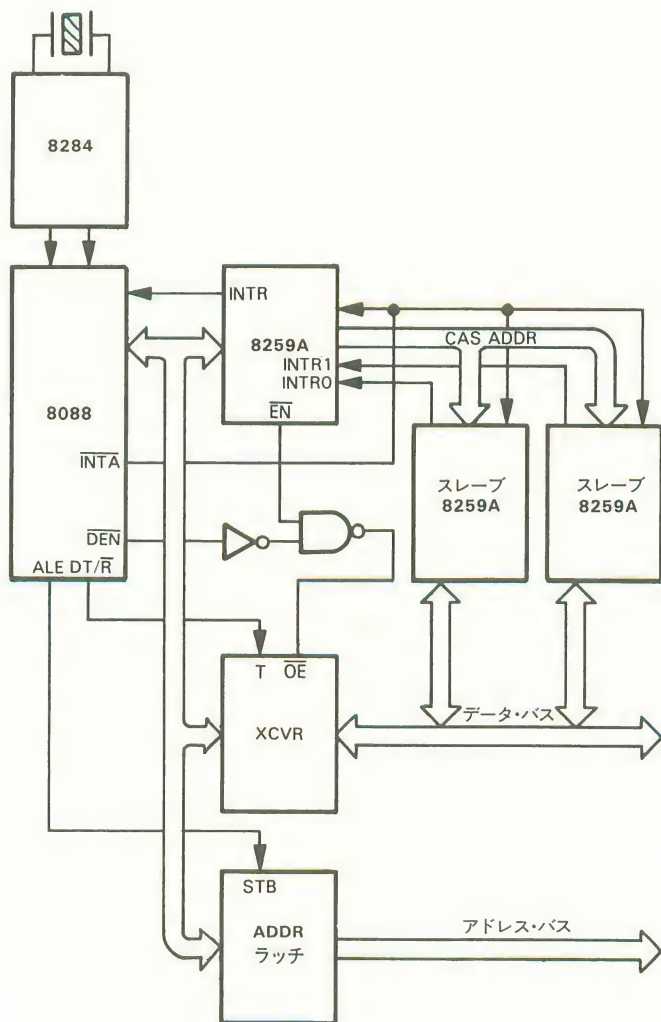


図8-32(a) ミニマム・モード8086に接続された8259—多重化バス

プ・ナンバーを供給しなければならないならば、8086はデータ・バス・トランシーバをイネーブルにする。

EN出力は通常ハイで、8086/8288がバス・トランシーバをコントロールすることを可能にしている。スレーブ・インタラプト処理時の適当なスレーブを選択するために、マス





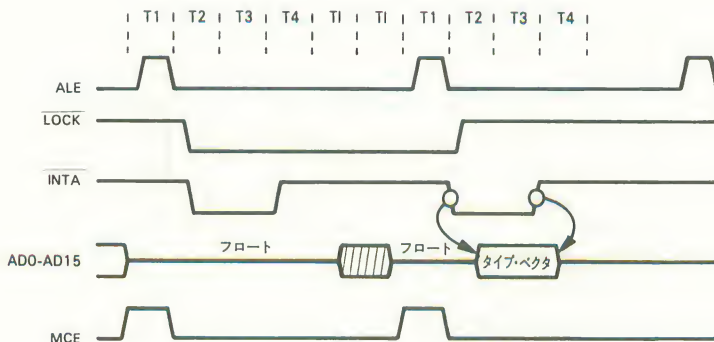


図8-34 8086ローカル・バス上に8259AがCASを出力するタイミング

## 8.6 8086バス・タイミング図の解釈

8086のミニマム・モードとマキシマム・モードのバス・タイミング図が、この本の後部のデータ・シートに示されている。このタイミング図は、次のように6つの区分に分けられる。

1. アドレスとALEのタイミング
2. リード・サイクルのタイミング
3. ライト・サイクルのタイミング
4. インタラプト・アクノリッジのタイミング
5. レディのタイミング
6. バス・コントロール移動のタイミング

信号のA.C.特性はCPUクロックに関して指定されているので、信号の大多数の間の関係は単に、信号が相対的な位置にあるクロックのエッジを識別しているクロック期間を決めて、適当な最小/最大のパラメータの値を加えるか減じることによって、推測することができる。この方法で補正できないシステム・タイミングの1つは、最小と最大のパラメータの値の間の“ワースト・ケース”の関係である（これはトラッキング関係としても知られている）。

たとえば、指定された最小と最大のターン・オンとターン・オフのディレイを持つ信号を考える。素子の特性に依存して、ワースト・ケースの解析が可能性を示したとしても、構成要素が同時に最大のターン・オンと最小のターン・オフのディレイを示す可能性はない。

この議論は、MOS素子の特性であり、したがって8086のA.C.特性に適用できる。ここで付言すると、最小と最大のディレイ・パラメータ混合のワースト・ケース解析は一般に、得られるワースト・ケースを超えていることがあげられる。したがって、より悪いワースト・ケースの値を得るために、解析による値を主観的にさらに下げることはできない。次に、トラッキング関係の影響を受けやすい8086のタイミングの特定の範囲に対するガイドラインを検討する。

## 8.7 ミニマム・モード・バスのタイミング

### 8.7.1 アドレスとALE

多重化バスから有効なアドレスを得る素子の能力を決定するために、アドレス/ALEのタイミング関係は重要である。8282と8283のラッチはALEの立下りエッジでアドレスを捕えるので、きわどいタイミングにはALEが終了するときのアドレス・ラインの状態が含まれる。パラメータ  $T_{AVAL} = T_{CLCH} - 60\text{ns}$  は、ALEの立下りエッジの58ns前で、CPUでのアドレスは有効であることを保証している。

これは、8282/8283で必要とされるストロブの終わりに対する0のデータ・セットアップ・タイムを満たし、有効なアドレスが捕えられることを確実にしている。アドレスはTLLAXパラメータによってALEの終わりを過ぎても有効であることが保証されている。最新の可能性のあるALEによってアドレスは有効でないことを意味するように考えられるが、この仕様はTCHLLとTCLAXの関係が無効にする。

TLLAXのタイミングは、タイミング図でA19-A16が示されているだけであるが、すべてのアドレス・バスに適用される。アドレスについてのTCLAX<sub>min</sub>の仕様は、遅いALEによって制限されなくても、バスがフロート状態となる最も速い可能性のある時間を示している。TCLAXは、リード・サイクルの間の多重化アドレス/データ・ラインAD15-0に適用されるだけである。

ライト・サイクルの間、多重化アドレス/データ・バスは直接にアドレスからライト・データに切り替わる。ALEに対するアドレス・ホールド・タイムは、TCLAXで指定される絶対最小値(ALE終結が速い場合)で、TLLAXの仕様でも保証されている。読み込みと書き込みの両方の場合に対して、書き込みの場合の多重化アドレス/データ・バスと同じタイミングで、A19-A16のラインは直接にアドレスからステータスに切り替わる。最小のALEパルス幅はTLHLL<sub>min</sub>で保証され、これはTCLLH<sub>max</sub>とTCHLL<sub>min</sub>の関係から得られる値よりも大きい。分離アドレス・バス上での有効なアドレスに対する最悪のディレイを決めるためには、次の2つの経路を考慮しなければならない。

1. 有効アドレスのディレイ
2. ALEのディレイ

8282と8283はラッチを用いているので、ALEがアクティブとなるまで有効アドレスはアドレス・バスに転送されない。アドレスが有効となるまでのディレイTCLAV<sub>max</sub>とALEがアクティブとなるまでのディレイTCLLH<sub>max</sub>の比較から、TCLAV<sub>max</sub>がワースト・ケースであることが示される。ラッチ伝播ディレイの減算によって、バス・サイクルの始まりからのワースト・ケースのアドレス・バスが有効となるまでのディレイが得られる。

### 8.7.2 リード・サイクルのタイミング

リード・サイクルのタイミングは次の3つの部分から成る。

1. バスの条件付け



2. リード・コントロール信号のアクティブ化
3. データ・トランシーバのイネーブルと方向制御の設定

メモリまたはI/O素子が直接に多重化アドレス/データ・バスに接続されていれば、T<sub>AZRL</sub>パラメータは、リード・コントロールがアクティブとなり選択された素子がバスを駆動できるようになる前に、8086がバスをフロート状態にすることを保証している。バス・サイクルの終わりで、次のバス・サイクルに対してアドレスを駆動する8086との競合を避けるならば、TRHAVパラメータは選択された素子が満たすべきバス・フロート・ディレイを指定している。次のバス・サイクルは、T<sub>4</sub>に続くCLK期間あるいはいくつかのCLK期間の後に開始される。

CPUにおいてリード・アクティブからデータが有効になるまでの最小ディレイは、 $2T_{CLCL} - T_{CLRL_{max}} - T_{DVCL} = 205ns$ である。最小パルス幅は、325nsの最小パルス幅を与える $2T_{RLRH}$ である。

DT/ $\overline{R}$ はバス・サイクルの初期に確定し、それ以上の考慮は必要としない。

読み込みの間、 $\overline{DEN}$ 信号は、トランシーバが適当なデータ・セットアップ・タイムでCPUにデータを伝えるのを可能とし、必要なホールド時間の間はそれを続けなければならない。 $\overline{DEN}$ ターン・オン・ディレイは、8086によって要求されるデータが有効となる前に、 $T_{CLCL} + T_{CHCL_{min}} - T_{CVCTV_{max}} - T_{DVCL} = 129ns$ のトランシーバ・イネーブル・タイムを与える。

8086のデータ・ホールド・タイム $T_{CLDX_{min}}$ と最小の $\overline{DEN}$ ターン・オフ・ディレイ $T_{CVCTX_{min}}$ は同じクロック・エッジに対して共に10nsの相対値を持つので、ホールド・タイムは保証される。さらに、次のバス・サイクルに対するアドレスで8086がバスを駆動する前に、 $\overline{DEN}$ はトランシーバをディスエーブルにしなければならない。最大の $\overline{DEN}$ ターン・オフ・ディレイ( $T_{CVCTX_{max}}$ )を8086出力のアドレスに対する最小ディレイ( $T_{CLRH_{min}}$ )と比較すると、CPUが多重化バスをアドレスで駆動する少なくとも55ns前に、トランシーバはディスエーブルとなることが示される。

### 8.7.3 ライト・サイクルのタイミング

ライト・サイクルは次の3つの主要な機能から成る。

1. システムへのライト・データの供給
2. ライト・コマンドの発生
3. データ・バス・トランシーバのコントロール

ライト・データとライト・コマンドは共にT<sub>2</sub>の立上りエッジからイネーブルとなる。最小の $\overline{WR}$ アクティブ・ディレイ $T_{CVCTV_{min}}$ と最大のライト・データ・ディレイ $T_{CLDV}$ の比較から、ライトがアクティブになってから100ns後まではライト・データが有効ではないことが示される。したがって、システム中の素子は有効なデータを保証するために、ライト・コマンドの立上りエッジよりも立下りエッジでデータを捕えなければならない。HOLDまたは $\overline{RQ/GT}$ の入力によってバスから強制的に切り離されたときだけ、8086はライトの後にバスをフロート状態にし、そうでなければ、8086は単に次のバス・サイクルの開

始で出力の駆動をデータからアドレスに切り替える。リード・サイクルと同じく、次のバス・サイクルは  $T_4$  に続くクロック期間あるいはいくつかのクロック期間の後に開始される。

$\overline{\text{WRITE}}$  の立下りエッジの前、最小の  $2\text{TCLCL} - \text{TCLDV}_{\max} + \text{TCVCTX}_{\min} = 300\text{ns}$  で 8086 からのデータは有効である。最小の  $\overline{\text{WRITE}}$  パルス幅は  $\text{TWLWH} = 340\text{ns}$  である。ライト後  $\text{TWHDX}$ 、CPU は有効なデータを保持する。 $\text{TWHDX}$  の仕様は、 $\text{TCLCH}_{\min}$  と  $\text{TCHDX}_{\min}$  の関係から得られる結果を無効にする。これは、 $\overline{\text{WR}}$  後  $18\text{ns}$  だけライト・データが有効となることを示している。 $\text{TCHDX}$  の最小のバス・フロート・タイムは、 $\text{TCVCTX} + \text{TWHDX} < \text{TCLCH} + \text{TCHDX}$  のときだけ影響する。

トランシーバの方向のコントロール信号  $\text{DT}/\overline{\text{R}}$  は、各リード・サイクルの終わりで伝送が条件づけられ、ライト・サイクルの間は変化しない。これにより、多重化バス上のアドレスを壊すことなく、アドレスが有効な間の、サイクルの初期においてトランシーバ・イネーブル信号  $\overline{\text{DEN}}$  をアクティブとすることができる。選択された素子に対するデータ・ホールド・タイムを保証するために、ライト後に最小の  $\text{TCLCH}_{\min} + \text{TCVCTX}_{\min} - \text{TCVCTX}_{\max} = 18\text{ns}$  で  $\overline{\text{DEN}}$  はディスエーブルとなる。再び最小の  $\text{TCVCTX}$  を最大の  $\text{TCVCTX}$  で評価したので、トランシーバのディスエーブルに対するライトの終わりからの実際のデイレイは約  $60\text{ns}$  となる。

#### 8.7.4 インタラプト・アクノリッジのタイミング

インタラプト・アクノリッジ・シーケンスは、2つのインタラプト・アクノリッジ・バス・サイクルから成る。各サイクルのタイミングは、コントロール信号のタイミングとアドレス/データ・バスのタイミングを除いて、リード・サイクルのタイミングと同一である。

$\overline{\text{INTA}}$  コントロール信号は、 $\overline{\text{WR}}$  コントロール信号と同じタイミングを有する。8086 においてコントロールからデータ有効までの  $260\text{ns}$  のアクセス・タイムを与えるために、 $\overline{\text{INTA}}$  は  $T_2$  の開始の  $110\text{ns}$  以内はアクティブである。8086 のデータ・ホールド・タイムを満たすために、最小の  $\text{TCVCTX}_{\min} = 10\text{ns}$  に対して  $T_4$  の立上りエッジに続いて  $\overline{\text{INTA}}$  コントロールがアクティブとなる。これは最小の  $\overline{\text{INTA}}$  パルス幅が  $300\text{ns}$  であることを保証するが、信号デイレイ・トラッキング ( $\text{TCVCTX}_{\max} = 110$  のとき、 $\text{TCVCTX}_{\min} = 50$ ) を考慮に入れると、 $340\text{ns}$  の最小パルス幅となる。 $\overline{\text{INTA}}$  の最大インアクティブ・デイレイは  $\text{TCVCTX}_{\max} = 110\text{ns}$  で、8086 は次のクロック・サイクルに対して  $15\text{ns}$  ( $\text{TCLAV}_{\min}$ ) まではバスを駆動しないので、出力をフロート状態にするためにローカル・バス上のインタラプト素子は  $105\text{ns}$  を利用できる。データ・バスにバッファが用いられていれば、出力をフロート状態にするために  $\overline{\text{DEN}}$  はローカル・バス・トランシーバに同じ時間を与える。

$\text{TCLAZ}$  以内に  $\overline{\text{INTA}}$  サイクルの開始の  $T_1$  から、多重化アドレス/データ・バスはフロート状態になる。多重化アドレス/ステータスの上位4つのラインはフロートとはならない。 $A_{19}-A_{16}$  に関するアドレス値は不定であるが、ステータス情報は有効である ( $S_3 = 0$ ,  $S_4 = 0$ ,  $S_5 = \text{IF}$ ,  $S_6 = 0$ ,  $S_7 = \overline{\text{BHE}} = 0$ )。多重化アドレス/データ・ラインは、 $\overline{\text{INTA}}$  バス・サイクルの  $T_4$  に続くクロック期間までフロート状態のままである。このシーケンスは、2つの  $\overline{\text{INTA}}$  バス・サイクルに対して起きる。第2の  $\overline{\text{INTA}}$  バス・サ

イクルで 8086 によってリードされるインタラプト・ナンバーは、リード・サイクルのデータ・セットアップとホールドの時間を満たさなければならない。

$\overline{\text{DEN}}$  と  $\text{DT}/\overline{\text{R}}$  の信号は、各  $\overline{\text{INTA}}$  サイクルに対してイネーブルとなり、2つのサイクルの間でアクティブのままではない。これら2つの信号に対するタイミングは、 $\overline{\text{INTA}}$  とリード・バス・サイクルにおいて同一である。

### 8.7.5 レディのタイミング

8086READY信号と8284に入力されるシステム・レディ信号(RDY)の詳細なタイミングの必要条件は、この章の初めに示してある。RDYは一般に、選択された素子のアドレス・デコードあるいはアドレスとコントロール信号 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 、 $\overline{\text{INTA}}$  から生成される。

RDYがアドレス・デコードによってイネーブルになると、考慮すべき2つの場合が存在する。通常はノット・レディのシステムに対して、有効アドレスからレディを生成してウエート状態を挿入しないための時間は、 $2\text{TCLCL} - \text{TCLAV}_{\text{max}} - \text{TRIVCL}_{\text{max}} = 255\text{ns}$  である。この時間は、バッファのデレイと、選択された素子がウエート状態を必要としないでRDYラインをハイに駆動するかどうかを判断するためのアドレス・デコードに利用できる。

ウエート・クロック期間が必要ならば、ユーザのハードウェアは適当なレディ・デレイを備えていなければならない。アドレスは次のALEまで変化しないので、このバス・サイクル中、RDYは有効のままである。通常はレディのシステムでは、ウエート状態を必要とする選択された素子はまた、RDYラインをディスエーブルとするために255nsを有している。ユーザのハードウェアは、適当な数のウエート状態のクロック期間だけRDYを再びイネーブルとすることを遅らせなければならない。

RDYが $\overline{\text{RD}}$ コントロールによってイネーブルとなるならば、 $\text{TCLCL} - \text{TCLRL}_{\text{max}} - \text{TRIVCL}_{\text{max}} = 15\text{ns}$  は外部ロジックで利用可能である。 $\overline{\text{WR}}$  コントロールが用いられていれば、 $\text{TCLCL} - \text{TCVCTV}_{\text{max}} - \text{TRIVCL}_{\text{max}} = 55\text{ns}$  が利用可能である。

アドレスあるいはコントロール信号で生成されるRDYの比較から、アドレス・デコーディングが最も良いタイミングを与える。システムが通常はレディでなければ、ウエート状態を必要としない素子に対してRDYを供給するためには、単にアドレス・デコーディングが用いられる。一方、ウエート状態を必要とする素子は、ウエート状態を発生させるために、アドレス・デコードとコントロール信号の組合せを用いる。システムが通常はレディならば、ウエート状態を必要としない素子はRDYに対して何も行なわない。一方、ウエート状態が必要な素子は、アドレス・デコードによってRDYをディスエーブルとして、RDYが再びイネーブルとなるまで遅らせるためにアドレス・デコードとコントロール信号を用いなければならない。システムがメモリに対してウエート状態を必要とせず、すべてのI/O素子に対する $\overline{\text{RD}}$ と $\overline{\text{WR}}$ に固定した数のウエート状態が必要ならば、 $\text{M}/\overline{\text{IO}}$  信号は、ウエート状態のクロック期間が必要であることを前以て示すものとして用いることができる。これにより、アドレス・デコードをフィードバックさせることなしに、共通の回路でシステム全体のレディのタイミングをコントロールすることができる。



### 8.7.6 バス・コントロール移動のタイミング

詳細な HOLD/HLDA のタイミングについてはこの章の後で言及している。

$\overline{\text{TEST}}$  入力は、WAIT 命令の実行中にだけ、8086 によってサンプルされる。検出を確実にするために、 $\overline{\text{TEST}}$  信号は WAIT 命令の間、最小 6 クロックの期間はアクティブでなければならない。

## 8.8 マキシマム・モード・バスのタイミング

マキシマム・モードの 8086 バス動作は、論理的にミニマム・モードと同等である。詳細なタイミングの解析には、8086 CPU と 8288 バス・コントローラによって生成される信号が含まれる。

ミニマム・モードの 8086 によって与えられる供給信号に加えて、8288 はシステムの融通性を拡張する付加的なコントロール信号を備えている。以下の解説で、信号の関係を検討すれば、同等のミニマム・モードの有用性よりも、適切なマキシマム・モードの有用性を用いることは確実となる。

### 8.8.1 アドレスとALE

マキシマム・モードにおいて、アドレス情報は常に 8086 から得られるが、ALE ストローブは 8288 バス・コントローラによって作られる。ALE と有効アドレスとの間のワースト・ケースの関係を決定するためには、8086 からのステータス  $\overline{S0} - \overline{S2}$  に関して 8288 の ALE のアクティブについて解析する必要がある。

マキシマム・モードのタイミング図は、ALE 生成の 2 つの可能性のあるディレイの経路を示す。第 1 は TCHSV + TSVLH で、これは T1 に先行するクロック期間の立上りエッジから測定される。第 2 の経路は TCLLH で、これは T1 の開始から測定される。8288 はステータス・ラインをパッシブ状態 ( $\overline{S0}, \overline{S1}, \overline{S2} = 1, 1, 1$ ) にすることからバス・サイクルを初期化するので、クロックがハイの時間が最小 ( $\text{TCHCL}_{\min}$ ) の間に 8086 のステータスの発行 ( $\text{TCHSV}_{\max}$ ) が遅れると、T1 の開始までにステータスの変化は起きず、ALE はステータスが変化した後 TSVLH に発行される。T1 の開始より前にステータスが変化すると、8288 は T1 の開始後 TCLLH まで ALE を発行しない。結果としてワースト・ケースの ALE をイネーブルにするディレイ (T1 の開始に関して) は、 $\text{TCHSV}_{\max} + \text{TSVLH}_{\max} - \text{TCHCL}_{\min} = 58\text{ns}$  である。

ALE をイネーブルとするディレイを無視すれば、ALE の立下りエッジは 8288 において T1 の正のクロック・エッジでトリガされる。結果的に最小の ALE パルス幅は、 $\text{TCHL} - 0$  を仮定すれば  $\text{TCLCH}_{\max} - 58\text{ns} = 75\text{ns}$  である。 $\text{TCHCL}_{\min}$  は  $58\text{ns}$  の ALE イネーブル・ディレイを導びくと仮定されているので、 $\text{TCLCH}_{\max}$  を用いる必要がある。8288 あるいは 8283 においてアドレスを捕えるために、ALE の立下りエッジの前  $\text{TCLCH}_{\min} + \text{TCHLL}_{\min} - \text{TCLAV}_{\max} = 8\text{ns}$  はアドレスが有効であることが保証される。ここでは再び安全をみて  $\text{TCHLL} = 0$  を仮定した。アドレスと ALE は別の素子で駆動されるので、

A.C.特性のトラッキングは仮定できないことに注意。

ラッチに対するアドレス・ホールド・タイムはT1の終わりまでアドレスが有効であることによって保証され、一方ALEはT1において正のクロック遷移から最大15nsでディスエーブルになる( $TCHCL_{min} - TCHLL_{max} = 52ns$ のアドレス・ホールド・タイム)。アドレスからステータスとライト・データあるいはトライステート(リードに対して)への多重化バスの遷移は、ミニマム・モードのタイミングと同一である。また、アドレス有効のディレイ(TCLAV)は有効アドレス確定におけるクリティカルな経路のままなので、有効なデータとレディに対するアドレス・アクセス・タイムはミニマム・モード・システムと同じである。

### 8.8.2 リード・サイクルのタイミング

マキシマム・モード・システムは、8086と8288によって別々に発生される2つのリード信号を提供する。8086の $\overline{RD}$ 出力信号のタイミングはミニマム・モード・システムと同一であるが、8288によって発生させられるリード・コントロール信号のA.C.特性は非常に良い。したがって分離されてバッファを持つシステム・バス上の素子は、8288のリード・コントロール信号を用いるべきである。8086の $\overline{RD}$ 信号は、多重化バス上に直接位置する素子に用いられる。

以下の評価では、8288のリード・コントロール信号のタイミングについてだけ考慮する。

8288は、別々のメモリとI/Oのリード・コントロール信号( $\overline{IORC}$ と $\overline{MRDC}$ )を出力し、この2つは同じA.C.特性を持つ。このコントロール信号は、T2の開始後TCLMLに発行され、T4の開始後TCLMHに終結する。最小のコントロール・パルスの長さは $2TCLCL - TCLML_{max} + TCLML_{min} = 375ns$ である。8086における有効データのアクセス・タイムは $2TCLCL - TCLML_{max} - TDVCL_{max} = 335ns$ となる。8288はバッファを用いたデータ・バスを有するシステムのために設計されているので、コントロール信号 $\overline{IORC}$ と $\overline{MRDC}$ は8086が多重化バスをフロート状態にする前にイネーブルとなる。したがって、コントロール信号 $\overline{IORC}$ と $\overline{MRDC}$ は多重化バスに直接接続している素子で用いられてはならず、さもないと結果として、8086のバス・フロートと素子のターン・オンの間にバス競合が生じる。

データ・バス・トランシーバの方向制御はT1で確定される。トランシーバは $\overline{DEN}$ によってT2の正のクロック遷移までイネーブルとなる。これにより、 $TCLCH + TCVNV_{min} = 123ns$ の8086バス・フロート・ディレイを与え、 $TCHCL_{min} + TCVNV_{max} - TDVCL_{max} = 187ns$ の間、8086において有効データに対してトランシーバはアクティブとなる。 $\overline{DEN}$ とコントロールの信号は共に、T4に対して最小10nsは有効なので、8086のデータ・ホールド・タイムTCLDZは保証される。 $\overline{DEN}$ ディスエーブルの最大45ns( $TCVN X_{max}$ )は、次の8086のバス・サイクルの開始(同じクロック・エッジから最小215ns)までにトランシーバがディスエーブルとなることを保証している。T4の正のクロック遷移において、次のバス・サイクルでの可能性のある書き込み動作の準備のために、DT/ $\overline{R}$ は送信に戻る。システムのメモリとI/Oの素子はバッファのあるシステム・バス上に存在するの



で、次のバス・サイクルに対する素子が選択される（およそ 2TCLCL）前かあるいはトランシーバがバス上にライト・データを駆動する（およそ 2TCLCL）前に、その出力をフロート状態にしなければならない。

### 8.8.3 ライト・サイクルのタイミング

マキシマム・モードでは、8288 はメモリと I/O に対して標準のものと先行したコントロール信号 ( $\overline{\text{MWTC}}$ ,  $\overline{\text{AMWC}}$ ,  $\overline{\text{IOWC}}$ ,  $\overline{\text{AIOWC}}$ ) を供給する。アドバンスド・ライト・コントロール信号は、標準ライト・コントロール信号の前の 1 クロック期間を通してアクティブである。アドバンスド・ライト・コントロール信号のタイミングはリード・コントロール信号のタイミングと同一である。

アドバンスド・ライト・パルス幅は  $2\text{TCLCL} - \text{TCLML}_{\max} + \text{TCLMH}_{\min} = 375\text{ns}$  であり、一方、標準ライト・パルス幅は  $\text{TCLCL} - \text{TCLML}_{\max} + \text{TCLMH}_{\min} = 175\text{ns}$  である。選択された素子に対するライト・データのセットアップ・タイムは、8086 からのデータが有効となるディレイ (TCLDV)，またはトランシーバ・イネーブル・ディレイ (TCVNV) の関数となる。有効なライト・データに対するワースト・ケースのディレイは、 $\text{TCLDV} = 110\text{ns}$  からトランシーバの伝播ディレイを引いたものになる。これは、データがアドバンスド・ライト・コントロール信号の立上りエッジ後 100ns まで有効ではないが、標準ライト・コントロール信号の立上りエッジの前およそ  $\text{TCLCL} - \text{TCLDV}_{\max} + \text{TCLML}_{\min} = 100\text{ns}$  は有効であることを意味する。どちらのライト・コントロール信号の立下りエッジ前  $2\text{TCLCL} - \text{TCLDV}_{\max} + \text{TCLMH}_{\min} = 300\text{ns}$  も、データは有効である。

アドバンスド・ライト・コントロールに対するデータとコントロール信号のオーバーラップは 300ns であり、一方標準ライト・コントロールとのオーバーラップは 175ns である。ライト・コントロール後、最小  $\text{TCLCH}_{\min} - \text{TCLMH}_{\max} + \text{TCVNX}_{\min} = 85\text{ns}$  でトランシーバはディスエーブルとなり、8086 は最小  $\text{TCLCH}_{\min} - \text{TCLMH}_{\max} + \text{TCHDZ}_{\min} = 85\text{ns}$  の間は有効なデータを供給する。これはライト・コントロール後に 85ns のライト・データ・ホールドを保証する。トランシーバは、後続のリード・バス・サイクルのためにその方向を変える前、 $\text{TCLCL} - \text{TCVNX}_{\max} + \text{TCHDTL}_{\min} = 155\text{ns}$  ( $\text{TCHDTL} = 0$  と仮定) でディスエーブルとなる。

### 8.8.4 インタラプト・アクノリッジのタイミング

マキシマム・モードの  $\overline{\text{INTA}}$  シーケンスは論理的にミニマム・モードのシーケンスと同一である。両者のインタラプト・アクノリッジ・サイクルのトランシーバ・コントロール ( $\overline{\text{DEN}}$  と  $\text{DT}/\overline{\text{R}}$ ) と  $\overline{\text{INTA}}$  コントロールのタイミングは、リード・サイクルのトランシーバ・コントロールのタイミングと同一である。ミニマム・モード・システムと同じく、各  $\overline{\text{INTA}}$  バス・サイクルに対して T1 の立上りエッジから多重化アドレス / データ・バスはフロート状態となり、各  $\overline{\text{INTA}}$  サイクルの T4 の後まで 8086 によって駆動されない。

第 2 の  $\overline{\text{INTA}}$  サイクルの間に外部ハードウェアによって返されるベクタのセットアップとホールドのタイムは、リード・バス・サイクルに対するデータのセットアップとホールド

ドと同じである。インタラプト・ベクタを与える素子がローカル・バスに接続されていれば、8086のバス・フロートから $\overline{\text{INTA}}$ のコマンドがアクティブになるまで $\text{TCLCL} - \text{TCLAZ}_{\text{max}} + \text{TCLML}_{\text{min}} = 130\text{ns}$ が利用できる。ローカル・バス上の選択された素子は、 $\overline{\text{DEN}}$ がなお8288によって生成されているので、システム・データ・バスのトランシーバをディスエーブルとしなければならない。

8288がI/O (I/Oバス) モードでなければ、8288の $\text{MCE}/\overline{\text{PDEN}}$ 出力はMCE出力となる。この出力は $\overline{\text{INTA}}$ サイクルの間はアクティブで、T1の間のALE信号とオーバーラップしている。MCEはマスタ8259Aからのカスケード・アドレスを上位AD15-AD8ラインの3つに送出するゲートとして利用できる。またMCEによってALEがカスケード・アドレスをアドレス・ラッチにラッチすることが可能となる。

次にアドレス・ラインは、システム・バス上に位置するスレーブ8259Aに対してCASアドレス選択を供給するために用いられる（この方法の記述については図8-32を参照）。MCEは、各 $\overline{\text{INTA}}$ サイクルに対してステータスあるいはT1開始の15ns以内はアクティブである。最初の $\overline{\text{INTA}}$ サイクルで80nsまで8086がバスをフロート状態にすることは保証していないので、最初のサイクルの間にMCEは多重化バスでのCASラインをイネーブルにはしない。最初のMCEは、MCEと $\overline{\text{LOCK}}$ のゲートによって禁止できる。8086の $\overline{\text{LOCK}}$ 出力は最初の $\overline{\text{INTA}}$ サイクルのT2の間にアクティブとなり、第2の $\overline{\text{INTA}}$ サイクルのT2の間にディスエーブルとなる。MCEとの $\overline{\text{LOCK}}$ のオーバーラップにより、最初のMCEをマスクして、第2のMCEをカスケード・アドレスのローカル・バスへのゲートとすることができる。

8259Aは第2の $\overline{\text{INTA}}$ バス・サイクルまでカスケード・アドレスを供給しないので、情報は失なわれない。ALEと同じく、ALEの立下りエッジで75nsのCASアドレス・セットアップを可能とするために、MCEはT1の開始の58ns以内で有効であることが保証される。ラッチに対してデータ・ホールド・タイムを与えるために、ALE後 $\text{TCHCL}_{\text{min}} - \text{TCHLL}_{\text{max}} + \text{TCLMCL}_{\text{min}} = 52\text{ns}$ の間、MCEはアクティブである。

8288がI/Oモードに切り替えられていれば、MCE出力は $\overline{\text{PDEN}}$ となり、すべてのI/O参照は、分離システム・バス上ではなくローカル・バス上の素子として仮定される。 $\overline{\text{INTA}}$ サイクルはI/Oサイクルと考えられるので、すべての割り込みはローカル・システム・バスからのものと仮定され、カスケード・アドレスはシステム・アドレス・バス上には送出されない。さらに、システム・バス上ではI/Oの伝送が起きないので、 $\overline{\text{DEN}}$ 信号はイネーブルとならない。ローカルI/Oバスもまたトランシーバによるバッファを有するならば、このトランシーバをイネーブルとするために $\overline{\text{PDEN}}$ 信号が用いられる。 $\overline{\text{PDEN}}$ のA.C.特性はDENと同一であり、 $\overline{\text{PDEN}}$ はI/Oの参照でイネーブルとなり、DENは命令またはデータのメモリ参照でイネーブルとなる。各種のモードのシステムの意味は後の章で述べる。

### 8.8.5 レディのタイミング

アドレス有効のタイミングに基づいては、レディのタイミングはマキシマム・モードと

ミニマム・モードのシステムで同じである。8284 での 8288 のコントロール有効から RDY の有効までのディレイは、 $TCLCL - TCLML_{\max} - TRIVCL_{\min} = 130ns$  である。この時間は、ウエート状態のクロック期間が挿入される必要性を判断するために外部回路で用いられる。外部回路は、ウエート状態を挿入するために RDY をディスエーブルにするか、ウエート状態を避けるために RDY をイネーブルにしなければならない。 $\overline{INTA}$ 、すべてのリード・コントロール、そしてアドバンスド・ライト・コントロールはこのタイミングを与える。

標準のライト・コントロールは、RDY が有効となる後まで有効とはならない。標準のものアドバンスド・ライト・コントロールは、すべてのライト・バス・サイクルに対して 8288 によって発生させられるので、選択された素子が標準のライト・コントロールを用いても、RDY を示すためにアドバンスド・ライト・コントロールが用いられる。

### 8.8.6 その他の考察

$\overline{RQ}/\overline{GT}$  のタイミングについては、この章の後で言及する。

マキシマム・モードで考慮すべき唯一の信号はキュー・ステータス・ライン QS0 と QS1 である。この信号は、アイドルとウエートのクロック期間を含んで、各クロック期間の立上りエッジ（ハイからローへの遷移）で変化する。キュー・ステータスは、BIU の動作とは独立に、エグゼキューション・ユニットのステータスを表わす。外部ロジックは、クロック・パルスのローからハイの遷移でこのラインをサンプルする。サンプル時は、QS0 と QS1 の信号は以前のクロック期間におけるキュー動作を識別しており、したがって CPU の動作とは 1 クロック期間だけのずれがある。

$\overline{TEST}$  入力条件は、ミニマム・モードに対して述べたものと同一である。

## 8.9 バス・コントロールの移動 (HOLD/HLDA と $\overline{RQ}/\overline{GT}$ )

8086 自身とバス・マスタとして動作可能な他の素子の間で、ローカル・バスのコントロールを移すために用いられるプロトコル信号を 8086 は有している。ミニマム・モード構成は、8080A と 8085 のシステムと同一のシングル・レベルのハンドシェイクを与える。マキシマム・モード構成は、2 つのレベルの優先権でシステム構成を 2 つのレベルの交互のバス・マスタに拡張して、CPU のピンをより有効に用いるエンハンスド・パルス・シーケンス・プロトコルを有している。このプロトコル信号は 8086 ローカル・バスのコントロールを判定する。これをシステム・バスの判定と混同してはならない。

### 8.9.1 ミニマム・モード

ミニマム・モード 8086 システムでは、CPU に対するホールド・リクエスト入力 (HOLD) と CPU からのホールド・アックノリッジ出力 (HLDA) を用いる。ローカル・バスのコントロールを得るために、素子は CPU に対して HOLD を主張して、バスを駆動する前

に HLDA を待たなければならない。8086 がバスを放棄できれば、 $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{INTA}$  と  $M/\overline{IO}$  のコントロール・ライン、 $\overline{DEN}$  と  $DT/\overline{R}$  バス・コントロール・ライン、そして多重化アドレス/データ/ステータス・ラインをフロート状態にする。ALE 信号はフロートとはならない。CPU は HLDA でローカル・バスに対する要求の受け付けを通知する。これにより、要求している素子がローカル・バスのコントロールを得ることができる。

要求素子は、もうそれ以上ローカル・バスを必要としなくなるまで、HOLD リクエストをアクティブに維持しなければならない。8086 に対する HOLD リクエストは、直接にバス・インターフェイス・ユニットに影響を与え、間接的にエグゼキューション・ユニットに影響する。

エグゼキューション・ユニットは、さらに命令が必要となるかオペランドの転送が必要となるまで内部キューから実行を続ける。これにより、CPU と補助バス・マスタの動作の間に程度の小さいオーバーラップが可能となる。要求元のマスタが HOLD 信号を落とすと、8086 は HLDA を落として応答する。8086 はバスとコントロールの信号を再び駆動しない。この信号は、8086 がバス転送を行なう必要があるまでフロート状態を続ける。HOLD が落ちたときに 8086 は、なおその内部キューから実行しているのので、どの素子もバスを駆動していない期間が存在する。

バス・コントロールの遷移の間にコントロール・ラインが最小の VIH レベル以下にドリフトするのを防ぐために、22 キロオームのプルアップ抵抗をバス・コントロール・ラインに接続する必要がある。図8-35 のタイミング図は、CPU のクロックに関して、HOLD のサンプル、バスのフロート、HLDA のイネーブル/ディスエーブルのタイミングの 8086 におけるバス・コントロール・ハンドシェイク・シーケンスを示している。

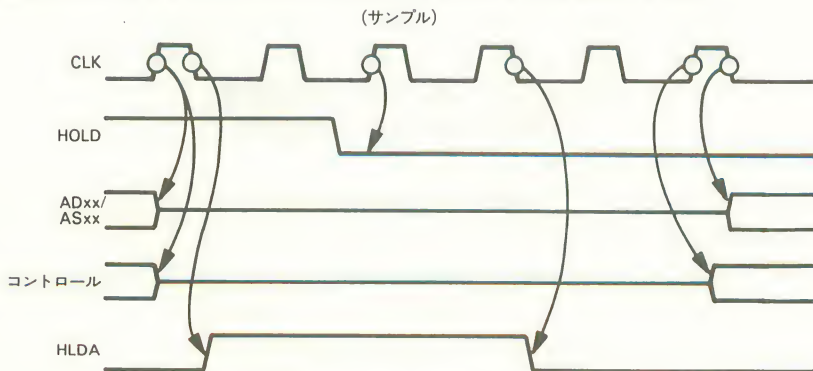


図8-35 HOLD/HLDAシーケンス

確実なシステム動作を確保するために、設計者は、要求元素子が 8086 のコントロール放棄の前にバスのコントロールを主張しないように、また、この素子が 8086 のバス駆動の前にバスのコントロールを放棄するように、保証しなければならない。HLDA と 8086 がバスをフロート状態とする間の最大ディレイは、 $TCHDZ_{\max} - TCHCL_{\min} - TCLHAV_{\min} = 10\text{ns}$  である。システムが 10ns のオーバーラップを許すことができなければ、8086 からの



HLDA のアクティブは素子に対してディレイを持たなければならない。

8086 がローカル・バス上のコントロール信号を駆動するまでの HOLD のインアクティブからの最小ディレイは、 $THVCH_{min} + 3TCLCL = 635ns$  であり、多重化バスの駆動のためには、このディレイは  $THVCH_{min} + 3TCLCL + TCHCL = 701ns$  となる。素子が指定された時間内にローカル・バスを解放しなければ、8086 に対する HOLD のインアクティブはディレイを持たなければならない。HLDA のインアクティブからバスの駆動までのディレイは、ローカル・バスのコントロール信号に対して  $TCLCL + TCLCH_{min} - TCLHAV_{max} = 158ns$ 、そしてローカル・データ・バスに対して  $TCLCL - TCLHAV_{max} = 240ns$  である。

#### (1) HOLD に対する HLDA の潜伏

HOLD リクエストに対する応答の決定は、バス・インターフェイス・ユニットによって行なわれる。この決定に影響する主要な要因には、現在のバス動作、CPU 内部の  $\overline{LOCK}$  信号の状態（ソフトウェアの LOCK プレフィックスによってアクティブになる）、そして保留状態のインタラプトがある。

$\overline{LOCK}$  がアクティブでなく、インタラプト・アクノリッジ・サイクルが進行中でなく、そして HOLD リクエストが受信されたときに BIU（バス・インターフェイス・ユニット）が T4 あるいは T1 のクロック期間を実行していれば、HLDA に対する最小の潜伏期間は次のようになる。

|        |                         |
|--------|-------------------------|
| 35 ns  | THVCH min (ホールド・セットアップ) |
| 65 ns  | TCHCL min               |
| 200 ns | TCLCL (バス・フロート・ディレイ)    |
| 10 ns  | TCLHAV min (HLDA ディレイ)  |

-----  
310 ns @ 5 MHz

上記条件での HLDA に対する最大の潜伏期間は次のようになる。

|        |                      |
|--------|----------------------|
| 34 ns  | (セットアップ・タイム)         |
| 200 ns | 次のサンプルに対するディレイ       |
| 82 ns  | TCHCL max            |
| 200 ns | TCLCL (バス・フロート・ディレイ) |
| 160 ns | TCLHAV max (HLDA)    |

-----  
677 ns @ 5 MHz

ホールド・リクエストが受信されたときに、ちょうど CPU がバス・サイクルの初期化を行なっていれば、ワースト・ケースの応答時間は次のようになる。

|        |                        |
|--------|------------------------|
| 34 ns  | THVCH                  |
| 82 ns  | TCHCL max              |
| 7*200  | バス・サイクル実行              |
| N*200  | N個のウェート状態のバス・サイクル      |
| 160 ns | TCLHAV max (HLDA ディレイ) |

-----  
1.676  $\mu s$  @ 5 MHz, ウェート状態なし



ホールド・リクエストのミスに対する 200ns はバス・サイクル実行のディレイに含まれていることに注意。オペランドの転送が奇数バイト境界に対するワード転送ならば、この転送のために 2 つのバス・サイクルが実行される。BIUはこの 2 つのバス・サイクルの間のホールド・リクエストを受け付けず、このタイプの転送は、前記最大潜伏期間をさらに 4 つのクロック期間と N 個のウェイト状態だけ延長する。バス・サイクルにウェイト状態がなければ、最大値は 2.476 マイクロ秒となる。

ミニマム・モードの 8086 にはハードウェアの  $\overline{\text{LOCK}}$  出力はないが、それでも命令の中にはソフトウェアの LOCK プレフィックスが含まれる。CPU は内部的に、マキシマム・モードの 8086 と同様に LOCK プレフィックスに反応する。したがって LOCK は、プレフィックスに続く命令が完了するまでホールド・リクエストを受け付けられない。その結果、CX を (BX) に加えてその結果を (BX) にストアする ADD (BX), CX などの、1 つ以上のメモリ参照を行なう命令は、メモリ参照の間に他のバス・マスタがバスのコントロールを獲得することなしに実行できる。 $\overline{\text{LOCK}}$  信号は命令実行より 1 クロック期間多い間アクティブなので、HLDA に対する最大潜伏時間は次のようになる。

|                                                       |                        |
|-------------------------------------------------------|------------------------|
| 34 ns                                                 | THVCH                  |
| 200 ns                                                | 次のサンプルに対するディレイ         |
| 82 ns                                                 | TCHCL max              |
| $(M+1)*200$ ns                                        | LOCK 命令実行              |
| 200 ns                                                | セットアップ HLDA (内部)       |
| 160 ns                                                | TCLHAV max (HLDA ディレイ) |
| -----                                                 |                        |
| $(M*200 \text{ ns}) + 876 \text{ ns} @ 5 \text{ MHz}$ |                        |
| M はロックされた命令実行のクロック数                                   |                        |

ホールド・リクエストがインタラプト・アクノリッジ・シーケンスの開始に行なわれると、HLDA に対する最大潜伏時間は次のようになる。

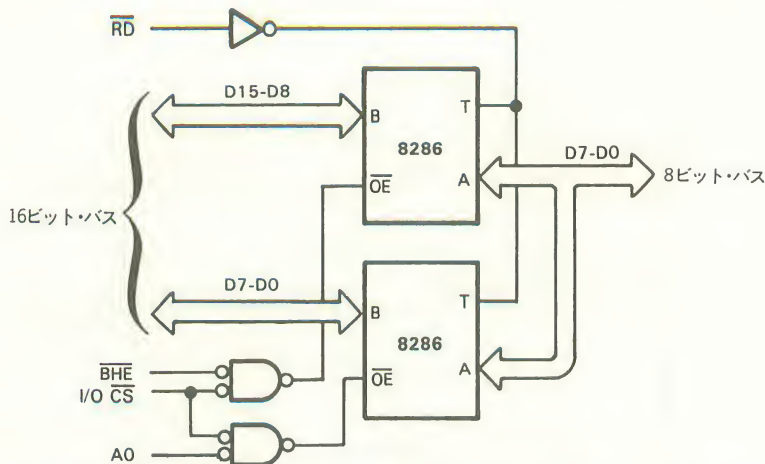
|                     |                     |
|---------------------|---------------------|
| 34 ns               | THVCH               |
| 82 ns               | TCHCL max           |
| 2600 ns             | INTA の 13 クロック・サイクル |
| 160 ns              | TCLHAV max          |
| -----               |                     |
| 2.876 $\mu\text{s}$ | @ 5 MHz             |

## (2) ミニマム・モードの DMA 構成

ミニマム・モードの HOLD/HLDA 信号の代表的な利用は、インテルの 8257-5 あるいは 8237 DMA コントローラなどの、DMA コントローラ素子とのバス・コントロールの交換である。図 8-36 は、8257-5 を用いたこのタイプの構成を機能的に示している。

DMA コントローラは 8086 のローカル多重化アドレス・データ・バスの上に位置し、8086 と A15-A8 の分離アドレス・ラッチを共有している。8257-5 のレジスタは、バスの上位を通してアクセスしなければならない。したがって、奇数アドレスのレジスタ ( $A0 = 1$ ) は奇数 I/O アドレスに対するバイト転送としてアクセスされ、偶数アドレスのレジスタはデータが上位バイトで転送されるものとしてワード I/O によってアクセスされる。80





86の読み込みと書き込みのコントロール信号は、8257-5の必要条件に適合する別々のI/Oとメモリのコントロールを得るために、分離しなければならない。8257-5からのAENコントロールで、8086のコントロール信号と、下位(A7-A0)と上位(A19-A16)のアドレス・バスのラッチをディスエーブルとすることが必要である。また、AENはA15-A8のアドレス・ラッチに対する8257-5のアドレス・ストロブ(ADSTB)を選択しなければならない。データ・バスにバッファが用いられていれば、DENラインのプルアップ抵抗はバッファをディスエーブルに保つ。DMAコントローラは、メモリとI/Oの間でバイトを転送するだけである。DMAコントローラは、以下に示されている16ビットから8ビットへのバス多重化回路から得られる8ビット・バス上にI/O素子が存在していることを必要とする。アドレス・ラインA7-A0は直接8257によって駆動され、BHEはA0を反転することによって生成される。A19-A16が用いられていれば、これは固定値あるいはソフトウェアで初期化される値を持つ付加的なポートによって与えられなければならない。この付加的ポートは、AENによってアドレスがイネーブルとなる必要がある。

図8-37は、システム・バスに接続されている8257を示す。

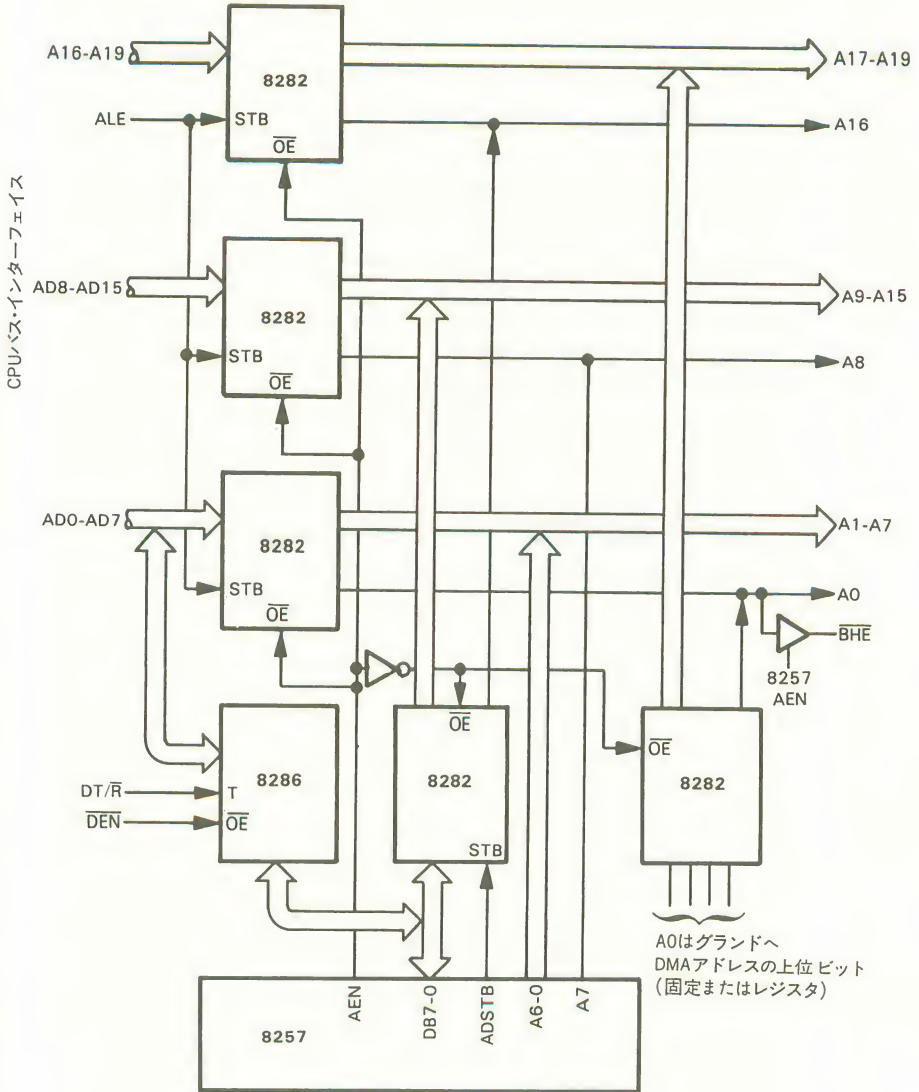
8257-5からの上位アドレスの保持に別のラッチを用いて、示されているように出力をアドレス・バスに接続すれば、16ビットのDMA転送が得られる。この構成において、AENはワード転送を可能とするためにA0とBHEを同時にイネーブルにする。それでもなお、AENはコントロールとアドレスのバスに対するCPUのインターフェイスをディスエーブルとしている。

### 8.9.2 マキシマム・モード ( $\overline{RQ}/\overline{GT}$ )

マキシマム・モード8086構成は、大きく異なるバス・コントロール移動のプロトコルをサポートしている。

#### (1) 共有システム・バス ( $\overline{RQ}/\overline{GT}$ の選択)

マキシマム・モードの $\overline{RQ}/\overline{GT}$ シーケンスは、8086と、全体的にローカル・バス上に位



コントロールは、データ・バスの操作が  
8ビット転送の構成と同じである。

図8-37 8086ミニマム・モード・システムの16ビット  
・データ転送のシステム・バスにおける8257

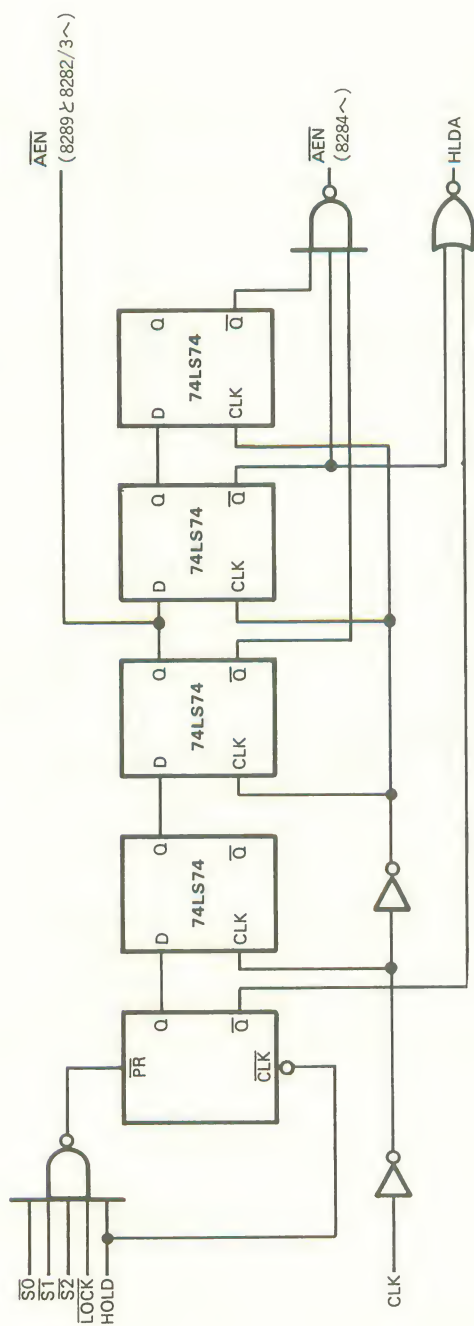


図8-38 マキシマム・モード 8086 の HOLD から  $\overline{AEN}$  のディスプレイ信号への変換



置してシステム・バスに対する完全なCPUのインターフェイスを共有するDMAコントローラなどのような別のバス・マスタとの間で、ローカル・バスのコントロールを移すために用いられる。システム・バスに対する完全なCPUインターフェイスには、アドレス・ラッチ、データ・トランシーバ、8288 バス・コントローラ、そして 8289 マルチマスタ・バス・アービタが含まれる。システムの別のバス・マスタが直接 8086 ローカル・バス上に位置していなければ、システム・バスの判定は必要であり、ローカル・バスの判定は必要でない。マルチマスタ・バス判定には 8289 バス・アービタが必要であり、 $\overline{RQ/GT}$  ロジックは用いられない。

簡単な HOLD/HLDA プロトコルを有する素子が、ちょうど1つのCPUが接続されているシステム・バスのコントロールを得るならば、図8-38の回路が用いられる。

実際この回路は、他のバス・マスタがホールド・リクエストを発行したときに、システム・バスからCPUを分離する簡単なバス・アービタである。8086 がアイドル状態 ( $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2} = 1$ ) を示し、 $\overline{LOCK}$  がインアクティブでホールド・リクエストがアクティブのとき、回路の出力  $\overline{AEN}$  (アクセス・イネーブル) は 8288 と 8284 をディスエーブルにする。 $\overline{AEN}$  がインアクティブになると、8288 はコントロール出力をフロート状態にして  $\overline{DEN}$  をディスエーブルとし、これによってデータ・バスのトランシーバはフロート状態になる。 $\overline{AEN}$  はまたアドレス・ラッチ (8282 または 8283) の出力をフロート状態にしなければならない。この動作でシステム・バスから 8086 が除去され、要求元素子のシステム・バスの駆動が可能となる。

8284 への  $\overline{AEN}$  信号は READY 入力をディスエーブルとして、8086 がシステム・バスのコントロールを取戻すまで待つために、8086 によってバス・サイクルの初期化を行なわせる。CPU はこの期間にローカル・バスを能動的に駆動する。

要求元素子は、8086 開始のバス・サイクル、ロックされた命令の実行、あるいはインタラプト・アクノリッジ・サイクルの間、システム・バスのコントロールを得られない。8086 からの  $\overline{LOCK}$  信号は、8086 のバス・コントロールの維持を保証するために、 $\overline{INTA}$  サイクルの間アクティブである。ミニマム・モード 8086 の HLDA 応答と異なり、奇数アドレス境界のワード・オペランド転送の連続したバス・サイクルの間に、要求元素子はバスのコントロールを得ることができる。要求元素子の特性に依存して、他の 74LS74 の出力の1つをその素子に対する HLDA を生成するために利用できる。これは、バスを用いる前にディレイを必要とする素子を接続するときに有用である。

システム・バスの動作が終了すると、そのバス・マスタはシステム・バスのコントロールを放棄して HOLD リクエストを落とさなければならない。 $\overline{AEN}$  がアクティブとなった後、アドレス・ラッチとデータ・トランシーバはイネーブルとなるが、8086 開始のバス・サイクルが保留状態ならば、最小 105ns のディレイまたは最大 275ns のディレイが経過するまで、8288 はコントロール・ラインを駆動しない。システムが通常はノット・レディならば、選択された素子が転送を終了すると 8086 に READY を返して、8284 の  $\overline{AEN}$  入力は直ちにイネーブルになる。

システムが通常はレディならば、標準バス・サイクルに等価なアクセス時間を与えるの

に十分長く  $8284 \overline{AEN}$  入力を遅らせなければならない。設計において 74LS74 ラッチは、HOLD を解放した後に別の素子がシステム・バスをフロートにするために最小 TCLCH を与える。また、8284 の READY 検出を可能にするまえに、2TCLCL のアドレス・アクセスと  $2TCLCL - TAEVCH_{\max}$  (8288 コマンド・イネーブル・ディレイ) のコントロール・アクセスも与える。HLDA が図8-38 に示されているように生成されるならば、HLDA を発行する前に 8086 がバスを解放するために TCLCL を利用でき、HLDA は HOLD を失うとすぐに落とされる。

マキシマム・モードの 8086 にインターフェイスするためのこの技法を用いた 8257-5 の回路構成は、図8-37 から導くことができる。8257-5 はアドレス・ライン A15-A8 のバッファのための自分自身のアドレス・ラッチを有する。8257-5 はアドレス・バス上へのラッチをイネーブルにするためにその  $\overline{AEN}$  出力を用いる。

この回路での HOLD から HLDA への最大潜伏期間は、HOLD が発行されたときのシステムの状態に依存している。アイドルのシステムで、最大潜伏期間は、NAND ゲートを通しての伝播ディレイと R/S フリップフロップの  $(TD1) + 2TCLCL + TCLCH_{\max} + 74LS74$  と  $74LS02$  の伝播ディレイ (TD2) である。ロックされた命令では、 $TD1 + TD2 + (M + 2) * TCLCL + TCLCH_{\max}$  となる。ただし、M はロックされた命令の実行に必要なクロック数である。インタラプト・アクノリッジ・サイクルに対して、潜伏期間は  $TD1 + TD2 + 9 * TCLCL + TCLCH_{\max}$  となる。

## (2) 共有ローカル・バス ( $\overline{RQ/GT}$ の使用法)

$\overline{RQ/GT}$  プロトコルは、1 つまたは 2 つの他の命令セット拡張プロセッサ (コープロセッサ) あるいは特殊な機能のプロセッサを、8086 のローカル・バスと直接接続することを可能とするために開発されたものである。8086 の  $\overline{RQ/GT}$  ピンは、バス・コントロール交換の完全なプロトコルをサポートする。

バス・コントロール交換シーケンスは、ローカル・バスのコントロールを得るための別のバス・マスタからのリクエスト、ローカル・バスが放棄されたことを示すための 8086 からの許可、そして終了時の別のバス・マスタからの解放パルスから構成される。2 つの  $\overline{RQ/GT}$  ピン ( $\overline{RQ/GT0}$  と  $\overline{RQ/GT1}$ ) には優先順位があり、 $\overline{RQ/GT0}$  は高い優先権を持つ。優先権は、どちらかに応答する前に両方のピンでリクエストを受けたときだけ意味を持つ。たとえば、 $\overline{RQ/GT1}$  でリクエストを受け、次いで  $\overline{RQ/GT1}$  の許可の前に  $\overline{RQ/GT0}$  でリクエストを受けると、 $\overline{RQ/GT0}$  は  $\overline{RQ/GT1}$  に対して優先権を持つ。しかし、 $\overline{RQ/GT1}$  がすでに優先権を認められていれば、 $\overline{RQ/GT0}$  でのリクエストは  $\overline{RQ/GT1}$  で解放パルスが受かるまで待つ必要がある。

バス・インターフェイス・ユニットでの要求 / 許可の相互作用シーケンスは HOLD/HLDA と類似している。8086 は、命令をフェッチするためかオペランドの処理のためにバス・サイクルを要求するまで、その内部キューからの命令の実行を続ける。しかし、8086 がバスを必要とする前に解放のパルスを受け取れば、バス・サイクルが必要となるまでバスを駆動しない。

リクエスト・パルスを受け取ると、8086 は多重化アドレス / データ・バス、 $\overline{S0}$ ,  $\overline{S1}$ ,

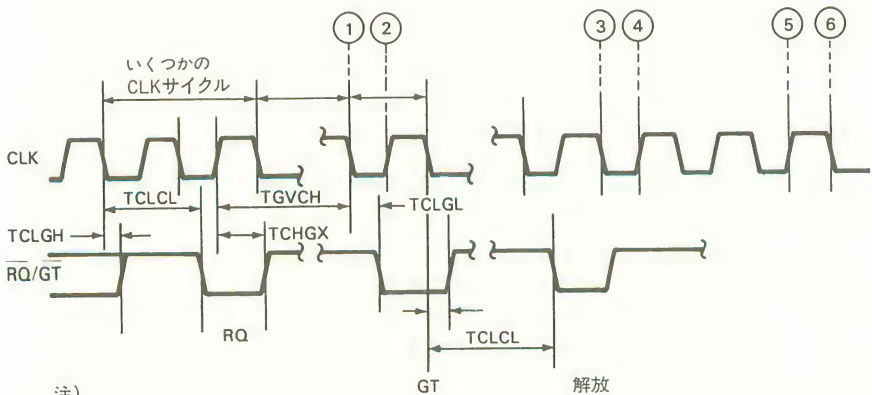
$\overline{S2}$  のステータス・ライン,  $\overline{LOCK}$  と  $\overline{RD}$  をフロートにする。この動作では, 8288 のコントロール出力をディセーブルとせず, またアドレス・ラッチをディセーブルともせず, アドレス・バスを駆動し続ける。8086 の出力がフロートの間にパッシブ・ステートを維持するために, 8288 は  $\overline{S0}$ ,  $\overline{S1}$ ,  $\overline{S2}$  のステータス・ラインに内部プルアップ抵抗を含む。このパッシブ・ステートは, データ・バスのバッファのトランシーバをイネーブルとするための 8288 のコントロール出力の開始, あるいは  $\overline{DEN}$  のアクティブ化を防止する。 $\overline{RQ}$  発行の素子が 8288 を用いないならば, 8288  $\overline{AEN}$  入力をディセーブルにすることによって 8288 のコントロール出力をディセーブルにしなければならない。また, 要求元素子によって用いられていないアドレス・ラッチは, 要求元素子によってディセーブルにされなければならない。

### (3) $\overline{RQ}/\overline{GT}$ の動作

$\overline{RQ}/\overline{GT}$  シーケンスの詳細なタイミングを図8-39に示す。

$\overline{RQ}/\overline{GT}$  ラインによってバス・コントロールの移動を要求するためには, 素子は 1 CPU クロック期間以上ラインをローに駆動する必要はない。これはリクエスト・パルスを構成する。 $\overline{RQ}/\overline{GT}$  ラインをサンプルする CPU のクロック・エッジに関して, 適当なセットアップとホールドのタイムを保証するために, リクエスト・パルスは CPU クロックと同期していなければならない。

リクエスト・パルスの発行後, 次のローからハイへのクロック・エッジから始まる, 許



注)

- ① 8086は, このエッジでパッシブ・ステートから $\overline{S2}$ ,  $\overline{S1}$ ,  $\overline{S0}$ をフロートにする。
- ② 8086は, このエッジでアドレス/ステータス/データ・バス,  $\overline{RD}$ と $\overline{LOCK}$ をフロートにする。
- ③ 他のマスターは, このエッジでパッシング・ステートから $\overline{S2}$ ,  $\overline{S1}$ ,  $\overline{S0}$ をフロートにする。
- ④ 他のマスターは, このエッジでアドレス/ステータス/データ・バス,  $\overline{BHE}$ , そして $\overline{LOCK}$ をフロートにする。
- ⑤ 8086は, コントロール・ラインを再び駆動する。
- ⑥ 8086は, アドレス/ステータス/データのラインを再び駆動する。

図8-39 要求 / 許可のシーケンス

可のパルスのサンプリングを素子が開始しなければならない。8086 はリクエストにすぐ続くクロック期間に許可のパルスで応答できるので、 $\overline{RQ}/\overline{GT}$ ラインはリクエストと許可のパルスの間に正のレベルに戻らない。したがって、エッジ・トリガ・ロジックは許可パルスを捕えることができない。また、リクエスト・パルスを発生する回路はCPUからの許可を検出する時間内にリクエストを取り除くことの保証が必要である。許可のパルスを受けた後、要求元素子はローカル・バスを駆動する。要求元マスタが許可を待つのを開始するのに用いる、アドレスまたはデータのバス、 $\overline{LOCK}$  または  $\overline{RD}$  をクロックのエッジまで8086 はフロートにしない。したがって、要求元マスタはローカル・バスを駆動する前に8086 のフロート・ディレイ・タイム (TCLAZ アドレス・フロートまたは TCHDZ データ・フロート) の間、待たなければならない。この予防策により、要求元マスタがローカル・バスのアクセスを得る間のバス競合を防止する。

8086 にローカル・バスのコントロールを返すために、別のバス・マスタは同じ  $\overline{RQ}/\overline{GT}$  ラインにリリース・パルスを出す。8086 は、リリース・パルスを検出して3クロック・サイクル後に  $\overline{S0} - \overline{S2}$  のステータス・ラインを駆動する。8086 は、ステータス・ラインが駆動されて TCHCL (クロックがハイの時間) の後にアドレス/データ・バスを駆動する。別のバス・マスタは、8086 がバスのコントロールを再び得るときまでに、ローカル・バスをフロート状態にして、8288 やアドレス・ラッチなどの他のインターフェイス回路を再びイネーブルとしなければならない。要求元素子は許可のパルスを受けて少なくとも1クロック・サイクル後までリリース・パルスを発行せず、以前のリリース・パルスの少なくとも1クロック・サイクル後まで新しいリクエストを発行してはならない。

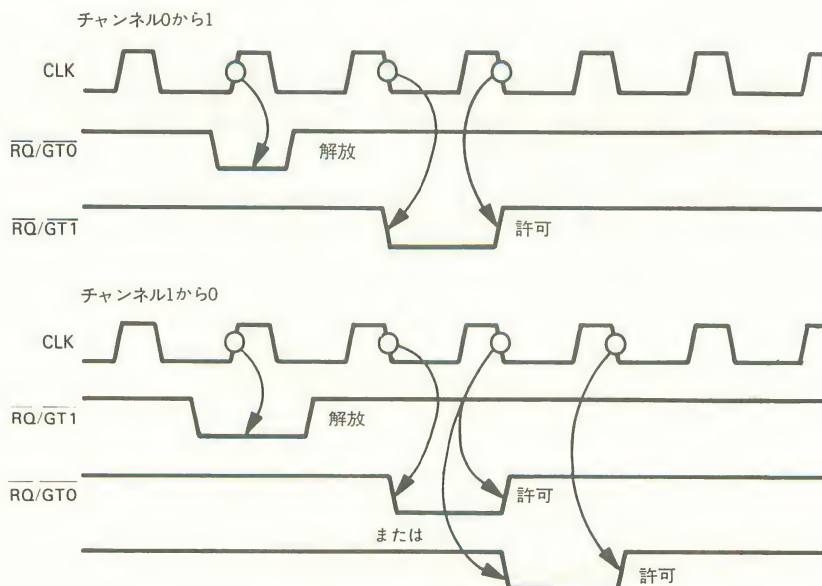


図8-40 チャンネル間のディレイ



(4)  $\overline{RQ} / \overline{GT}$ の潜伏

単一の $\overline{RQ} / \overline{GT}$ ラインでの $\overline{RQ}$ から $\overline{GT}$ へのディレイは、 $\overline{HOLD}$ から $\overline{HLDA}$ へのディレイに類似している。ミニマム・モードの8086に対して与えられたケースはまた、マキシマム・モードにも適用される。各ケースにおいて、8086による $\overline{RQ}$ の検出から要求元マスタによる $\overline{GT}$ の検出に対するディレイは、( $\overline{HOLD}$ から $\overline{HLDA}$ へのディレイ) - ( $T_{HVC}H + T_{CHCL} + T_{CLHAV}$ ) である。これは、アイドルのバス・インターフェイスでのクロック期間の最大ディレイである。他のすべての場合、ディレイはミニマム・モードの結果から476nsを引いたものに等しい。8086が前に $\overline{RQ} / \overline{GT}$ ラインの1つで許可を発行しているならば、最初の素子とその $\overline{RQ} / \overline{GT}$ ラインにリリース・パルスでインターフェイスを解放するまで、他の $\overline{RQ} / \overline{GT}$ ラインのリクエストは許可を受けられない。1つの $\overline{RQ} / \overline{GT}$ ラインの解放から他のラインの許可までのディレイは一般に、図8-40に示しているように1つのクロック期間である。

しばしば、 $\overline{RQ} / \overline{GT1}$ の解放から $\overline{RQ} / \overline{GT0}$ の許可までのディレイは、2つのクロック期間を要し、8086のエグゼキューション・ユニットからのコントロール移動のための保留状態のリクエストの関数となる。インターフェイスが他の $\overline{RQ} / \overline{GT}$ ラインでバス・マスタのコントロール下にあるとき、要求から許可までのディレイは他のバス・マスタの関数である。プロトコルには、8086が別のバス・マスタを強制的にバスから切り離せるよ

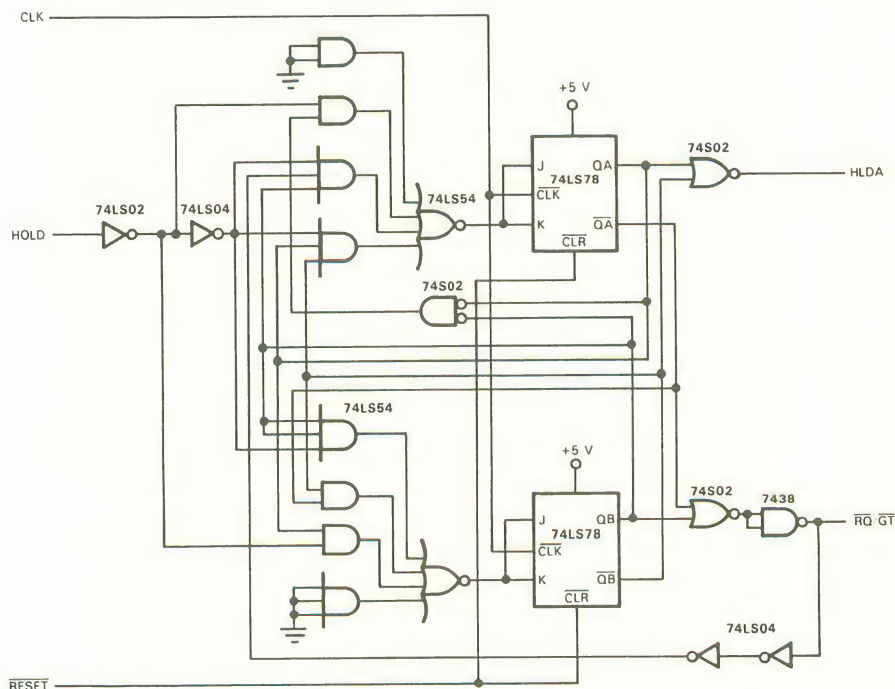


図8-41  $\overline{HOLD} / \overline{HLDA}$ から $\overline{RQ} / \overline{GT}$ への変換回路



うな機構は含まれていない。誤って別のバス・マスタがシステムをハング状態にしないことを確実にするためには、ウォッチドッグ・タイマを用いなければならない。

#### (5) $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ から $\overline{\text{RQ}}/\overline{\text{GT}}$ への変換

$\overline{\text{HOLD}}/\overline{\text{HLDA}}$  ハンドシェイク・シーケンスを  $\overline{\text{RQ}}/\overline{\text{GT}}$  パルス・シーケンスに変換する回路を図8-41に示す。

許可パルスを受けた後、8086がローカル・バスを切り離す  $\text{TCHCL}(\text{min})$  前に、 $\overline{\text{HLDA}}$  はイネーブルになる。要求元回路が  $\overline{\text{HLDA}}$  の 20ns 以内にバスを駆動するならば、1クロックの期間だけアクノリッジ・パルスを遅らせることが望ましい。 $\overline{\text{HLDA}}$  は、 $\overline{\text{HOLD}}$  がディスエーブルとなって1クロック期間以内に落とされる。 $\overline{\text{HLDA}}$  はまた、要求元マスタがステータス・ラインのコントロールを放棄するための  $2\text{TCLCL} + \text{TCLCH}$  と残りの信号をフロートとするための  $3\text{TCLCL}$  を与えるために、リリース・パルスの開始で落ちる。

# 第9章

## Multibus

Multibus は汎用マルチプロセッシング・システム・バスである。この標準的バスは、機械的、電氣的、そして形式の仕様を有する。Multibus は、インテルiSBC シングル・ボード・マイクロコンピュータの製品に用いられている。Multibus とコンパチブルな製品はまた、他の製造業者によっても提供される。マルチプロセッシング・システムを設計する人は、次の2つの重要な理由から Multibus を用いて自分のシステムを構成すべきである。

1. 新しいシステム開発に関連した時間とコストの節約のために。
2. Multibus に利用できる広範な種類の製品との互換性を得るために。

8086がマキシマム・モードで構成されるとき、8288バス・コントローラと8289バス・アービタは、Multibus システム・バスの電氣的特性とAC特性と完全に互換性のあるバス・アクセスとコントロールのインターフェイスを備えている。ミニマム・モードで構成されるとき、マルチプロセッサ・システムが必要でなければ（適当な信号をエンコードするための何か外部のロジックを有するが）、8086は Multibus で容易に動作できる。すべてのマルチプロセッサ・システムでは、8086はマキシマム・モードで構成される必要がある。

Multibus は、広範な種類の計算モジュールと同等に用いることの可能な、融通性のあるコミュニケーション・チャンネルを備えている。システムにおけるモジュールは、マスタあるいはスレーブである。マスタはバスの利用を得てデータ転送を開始するが、スレーブは単にデータ転送を行なう。バスは、システムで8ビットと16ビットのマスタとスレーブの混合を許している。バスは、16データ・ライン、20アドレス・ライン、8インタラプト・ライン、さらにコントロールとバス判定のラインをサポートしている。他のラインには、パワー・バス、パワー・バックアップ、そしてメモリをバッテリー・バックアップ・システムに切り替えるためのパワー・センス信号が含まれる。Multibus でのピン割当ての完全な一覧を表9-1と9-2に示す。以下に信号の機能的な記述を示す。

表 9-1 マルチバス・ボードP1コネクタのバス信号のピン割当て

|           | ピン | (素 子 側) |                   | ピン | (回 路 側) |                           |
|-----------|----|---------|-------------------|----|---------|---------------------------|
|           |    | ニーモニック* | 説 明               |    | ニーモニック  | 説 明                       |
| パワー・サブライ  | 1  | GND     | 信号 GND            | 2  | GND     | 信号 GND                    |
|           | 3  | +5V     | + 5 Vdc           | 4  | +5V     | +5Vdc                     |
|           | 5  | +5      | +5Vdc             | 6  | +5      | +5Vdc                     |
|           | 7  | +12V    | +12Vdc            | 8  | +12V    | +12Vdc                    |
|           | 9  | -5V     | -5Vdc             | 10 | -5V     | -5Vdc                     |
|           | 11 | GND     | 信号 GND            | 12 | GND     | 信号 GND                    |
| バス・コントロール | 13 | BCLK/   | バス・クロック           | 14 | INIT/   | イニシャライズ                   |
|           | 15 | BPRN/   | バス・プライオリティ入力      | 16 | BPRO/   | バス・プライオリティ出力              |
|           | 17 | BUSY/   | バス・ビジー            | 18 | BREQ/   | バス・リクエスト                  |
|           | 19 | MRDC/   | メモリ・リード・コマンド      | 20 | MWTC/   | メモリ・ライト・コマンド              |
|           | 21 | IORC/   | I/Oリード・コマンド       | 22 | IOWC/   | I/Oライト・コマンド               |
|           | 23 | XACK/   | トランスファー・アクノリッジ    | 24 | INH1/   | RAMディスエーブルのインビット 1        |
| バス・コントロール | 25 |         | 予備                | 26 | INH2/   | PROMまたはROMディスエーブルのインビット 2 |
|           | 27 | BHEN/   | バイト・ハイ・イネーブル      | 28 | AD10/   |                           |
|           | 29 | CBRQ/   | コモン・バス・リクエスト      | 30 | AD11/   |                           |
|           | 31 | CCLK/   | 一定クロック            | 32 | AD12/   | アドレス・バス                   |
|           | 33 | INTA/   | インタラプト・アクノリッジ     | 34 | AD13/   |                           |
| インタラプト    | 35 | INT6/   |                   | 36 | INT7/   |                           |
|           | 37 | INT4/   |                   | 38 | INT5/   |                           |
|           | 39 | INT2/   | パラレル・インタラプト・リクエスト | 40 | INT3/   |                           |
|           | 41 | INT0/   |                   | 42 | INT1/   | パラレル・インタラプト・リクエスト         |
| アドレス      | 43 | ADRE/   |                   | 44 | ADRF/   |                           |
|           | 45 | ADRC/   |                   | 46 | ADRD/   |                           |
|           | 47 | ADRA/   |                   | 48 | ADRB/   |                           |
|           | 49 | ADR8/   | アドレス・バス           | 50 | ADR9/   |                           |
|           | 51 | ADR6/   |                   | 52 | ADR7/   |                           |
|           | 53 | ADR4/   |                   | 54 | ADR5/   |                           |
|           | 55 | ADR2/   |                   | 56 | ADR3/   |                           |
|           | 57 | ADRO/   |                   | 58 | ADR1/   |                           |
| データ       | 59 | DATE/   |                   | 60 | DATF/   |                           |
|           | 61 | DATC/   |                   | 62 | DATD/   |                           |
|           | 63 | DATA/   |                   | 64 | DATB/   |                           |
|           | 65 | DAT8/   | データ・バス            | 66 | DAT9/   |                           |
|           | 67 | DAT6/   |                   | 68 | DAT7/   |                           |
|           | 69 | DAT4/   |                   | 70 | DAT5/   |                           |
|           | 71 | DAT2/   |                   | 72 | DAT3/   |                           |
|           | 73 | DAT0/   |                   | 74 | DAT1/   |                           |
| パワー・サブライ  | 75 | GND     | 信号 GND            | 76 | GND     | 信号 GND                    |
|           | 77 |         | 予備                | 78 |         | 予備                        |
|           | 79 | - 12V   | - 12Vdc           | 80 | - 12V   | - 12Vdc                   |
|           | 81 | +5V     | +5Vdc             | 82 | +5V     | +5Vdc                     |
|           | 83 | +5V     | +5Vdc             | 84 | +5V     | +5Vdc                     |
|           | 85 | GND     | 信号 GND            | 86 | GND     | 信号 GND                    |

すべてのニーモニックは Intel Corporation が 版權を所有 (1978)

すべてのニーモニックはIntel Corporation が 版權を所有 (1978)

\* 負が真 (アクティブ・ロー) の信号に対してこの本では、信号名の上に線を引くかあるいは信号名の後に斜線を引く (例 BUSY=BUSY/) 2つの記号を用いている。

表9-2 オプショナル・バス信号のP2コネクタピン割当て

| ピン | (素 子 側)          |                   | ピン | (回 路 側)           |                   |
|----|------------------|-------------------|----|-------------------|-------------------|
|    | ニーマニック           | 説 明               |    | ニーマニック            | 説 明               |
| 1  | GND              | 信号                | 2  | GND               | 信号GND             |
| 3  | V <sub>CCB</sub> | +5Vバッテリー          | 4  | V <sub>CCB</sub>  | +5V バッテリー         |
| 5  |                  | 予備                | 6  | V <sub>CCPP</sub> | +5V パルス化パワー       |
| 7  | V <sub>BBB</sub> | -5Vバッテリー          | 8  | V <sub>BBB</sub>  | -5V バッテリー         |
| 9  |                  | 予備                | 10 |                   | +                 |
| 11 | V <sub>DDB</sub> | +12Vバッテリー         | 12 | V <sub>DDB</sub>  | +12V バッテリー        |
| 13 | PFSR/            | パワー・フェイル・センス・リセット | 14 | 予備                | +                 |
| 15 | V <sub>AAB</sub> | -12Vバッテリー         | 16 | V <sub>AAB</sub>  | -12V バッテリー        |
| 17 | PFSN/            | パワー・フェイル・センス      | 18 | ACLO              | AC ロー             |
| 19 | PFIN/V           | パワー・フェイル・インタラプト   | 20 | MPRO/             | メモリ・プロテクト         |
| 21 | GND              | 信号GND             | 22 | GND               | 信号 GND            |
| 23 | +15V/            | +15V              | 24 | +15V              | +15V              |
| 25 | -15V/V           | -15V              | 26 | -15V              | -15V              |
| 27 | PAR1/            | パリティ 1            | 28 | HALT              | バス・マスタ HALT       |
| 29 | PAR2/            | パリティ 2            | 30 | WAIT/             | バス・マスタ WAIT STATE |
| 31 |                  |                   | 32 | ALE               | バス・マスタ ALE        |
| 33 |                  |                   | 34 | 予備                |                   |
| 35 |                  |                   | 36 | 予備                |                   |
| 37 |                  |                   | 38 | AUX RESET/        | リセット・スイッチ         |
| 39 |                  |                   | 40 |                   |                   |
| 41 |                  |                   | 42 |                   |                   |
| 43 |                  |                   | 44 |                   |                   |
| 45 | 予備               |                   | 46 |                   |                   |
| 47 |                  |                   | 48 | 予備                |                   |
| 49 |                  |                   | 50 |                   |                   |
| 51 |                  |                   | 52 |                   |                   |
| 55 |                  |                   | 56 |                   |                   |
| 57 |                  |                   | 58 |                   |                   |
| 59 |                  |                   | 60 |                   |                   |

注)

1. PFINはスレーブ・モジュール上でもし可能ならば、P1のINT0/への接続を選択できるべきである。
2. 未定義のピンはすべて将来のための予備である。

すべてのニーマニックは Intel Corporation が版權を所有 (1978)。

## 9.1 初期化信号ライン

### (1) $\overline{\text{INIT}}$

初期化信号は、前もって決められた状態にシステム全体をリセットする。 $\overline{\text{INIT}}$ は、バス・マスタの1つあるいは外部ロジックによって供給される。

## 9.2 アドレスとインヒビットのライン

### (1) $\overline{\text{ADR0}} - \overline{\text{ADR13}}$

20のアドレス・ラインが、アクセスされるメモリ位置あるいはI/Oポートのアドレスを

伝送するために用いられる。 $\overline{\text{ADR13}}$ は最上位ビットであり、 $\overline{\text{ADR0}}$ は最下位ビットである。8ビットのバス・マスタは、メモリのアドレス指定に16のアドレス・ライン ( $\overline{\text{ADR0}} - \overline{\text{ADR7}}$ ) を、I/Oポートの選択に8のアドレス・ライン ( $\overline{\text{ADR0}} - \overline{\text{ADR7}}$ ) を用いる。16ビットのバス・マスタは、20のアドレス・ラインでメモリのアドレス指定を、下位12のアドレス・ライン ( $\overline{\text{ADR0}} - \overline{\text{ADRB}}$ ) でI/Oポートの選択を行なう。しかし8088は、8ビット・バスのCPUと考えられるが、20のアドレス・ラインすべてを用いることができる。

#### (2) $\overline{\text{INH1}}$

インビットRAM信号は、RAMメモリ素子がアドレス・バス上のアドレスに応答するのを防止する。 $\overline{\text{INH1}}$ によって、ROMとRAMのメモリが同じメモリ領域に割り当てられているときに、ROMメモリ素子がRAM素子を無効にすることができる。

#### (3) $\overline{\text{INH2}}$

インビットROM信号は、ROMメモリ素子がアドレス・バス上のアドレスに応答するのを防止する。 $\overline{\text{INH2}}$ によって、ROMと補助ROMのメモリが同じメモリ領域に割り当てられているときに、補助ROMがROM素子を無効にすることができる。

$\overline{\text{INH1}}$ と $\overline{\text{INH2}}$ はまた、メモリ・マップドI/O素子がRAMとROMの素子をそれぞれ無効にできるようにするためにも用いられる。

#### (4) $\overline{\text{BHEN}}$

$\overline{\text{BHEN}}$ は、データがMultibusの上位8のデータ・ラインで伝送されることを示すのに用いられる。この信号は、16ビットのメモリまたはI/Oモジュールを用いるシステムで利用される。

## 9.3 データ・ライン

#### (1) $\overline{\text{DAT0}} - \overline{\text{DATF}}$

16の両方向データ・ラインが、メモリ位置あるいはI/Oポートとの情報交換に用いられる。 $\overline{\text{DATF}}$ は最上位ビットであるが、8ビットのシステムでは $\overline{\text{DAT0}} - \overline{\text{DAT7}}$ のラインだけが用いられ、 $\overline{\text{DAT7}}$ が最上位ビットになる。 $\overline{\text{DAT0}}$ は常に最下位ビットである。

## 9.4 バス競合解決ライン

#### (1) $\overline{\text{BCLK}}$

バス・クロックの負のエッジは、バス競合の同期化に用いられる。 $\overline{\text{BCLK}}$ はCPUクロックとは非同期である。 $\overline{\text{BCLK}}$ は、遅くなったり、停止したり、あるいはデバッグの間でシングル・ステップになる。 $\overline{\text{BCLK}}$ はCPUクロックとは非同期である。

#### (2) $\overline{\text{CCLK}}$

一定クロックは、特に示されていない一定周波数のクロック信号を供給する。

#### (3) $\overline{\text{BPRN}}$

バス・プライオリティ入力信号は、より高い優先権の素子がシステム・バスの使用を要



求していないことをバス・マスタに通知する。 $\overline{\text{BPRN}}$ は $\overline{\text{BCLK}}$ と同期している。この信号は、シリアルの優先権判定を用いるならば、“ディジィ・チェーン”となる。パラレルの優先権判定を用いるときは、バス・アービタが $\overline{\text{BPRN}}$ を生成する。

#### (4) $\overline{\text{BPRO}}$

これはバス・プライオリティ出力信号である。 $\overline{\text{BPRN}}$ と同様に、シリアルの優先権判定が用いられるときは“ディジィ・チェーン”となり、 $\overline{\text{BPRO}}$ は次に低い優先権のモジュールの $\overline{\text{BPRN}}$ 入力に加えられる。パラレルの優先権判定を用いるときは、バス・アービタがこの信号を供給しなければならない。 $\overline{\text{BPRO}}$ は $\overline{\text{BCLK}}$ と同期している。

#### (5) $\overline{\text{BUSY}}$

バス・ビジィ信号は、システム・バスが使用されていることを示すために、現在のバス・マスタによって供給される。 $\overline{\text{BUSY}}$ は、システム・バスのコントロールを得られるかどうかを判断するために、他の素子によって用いられる。 $\overline{\text{BUSY}}$ は $\overline{\text{BCLK}}$ と同期している。

#### (6) $\overline{\text{BREQ}}$

バス・リクエスト信号は、バス・マスタになることの要求を示すために、素子によって用いられる。 $\overline{\text{BREQ}}$ は $\overline{\text{BCLK}}$ と同期している。

#### (7) $\overline{\text{CBRQ}}$

$\overline{\text{CBRQ}}$ は、現在のバス・マスタに他のマスタのバス使用の要求を知らせるために、可能性のあるすべてのバス・マスタによって用いられる。 $\overline{\text{CBRQ}}$ がハイならば、現在のバス・マスタは、他の素子がバスを要求していなくて、現バス・マスタがバスを保持すべきであることを知る。

## 9.5 情報伝送プロトコルのライン

システム・バスのコントロールを有するバス・マスタは、データ伝送コントロール信号のすべてを発生する。すべてのアドレス信号（書き込みが行なわれるときはデータ信号も）伝送コントロール信号パルスの少なくとも50ns前で安定でなくてはならず、コントロール信号パルスが取り除かれてから少なくとも50ns後までの間は有効でなければならない。

情報伝送プロトコルのラインは、 $\overline{\text{BCLK}}$ と同期していない。

#### (1) $\overline{\text{MRDC}}$

メモリ・リード・コントロールは、メモリ位置のアドレスがアドレス・ライン上に設定されていて、そのアドレス位置の内容がデータ・ライン上に設定されるべきであることを示す。

#### (2) $\overline{\text{MWTC}}$

メモリ・ライト・コントロールは、メモリ位置のアドレスがアドレス・ライン上に設定されていて、データがシステム・データ・ライン上に設定されていることを示し、このデータはアドレス指定されたメモリ位置に書き込まなければならない。

#### (3) $\overline{\text{IORC}}$

I/O リード・コントロールは、入力ポートのアドレスがシステム・アドレス・ライン上に設定されていて、入力ポートのデータがデータ・ライン設定されるべきであることを示す。

(4)  $\overline{\text{IOWC}}$ 

I/O ライト・コントロールは、出力ポートのアドレスがシステム・アドレス・ライン上に設定されていて、データがシステム・データ・ライン上に設定されていることを示し、このデータはアドレス指定されたポートに出力されなければならない。

(5)  $\overline{\text{XACK}}$ 

すべての交換にはハンドシェイクを伴う。したがって、選択されたバス・スレーブは移動コントロール信号に応じて、バス・マスタにアクノリッジ信号を供給しなければならない。トランスファ・アクノリッジ信号は、指定された動作の完了を示すための必要な応答である。

(6)  $\overline{\text{AACK}}$ 

アドバンスド・アクノリッジ信号は、8080A マイクロプロセッサで用いられる。 $\overline{\text{AACK}}$  は、CPU がウェイト状態に入ることなく指定された動作の終了を可能とする先行のアクノリッジである。 $\overline{\text{AACK}}$  を供給するバス・スレーブはまた、 $\overline{\text{XACK}}$  も供給しなければならない。この必要条件は、すべてのバス・マスタが  $\overline{\text{AACK}}$  信号に応答するとは限らないので、満たされる必要がある。

## 9.6 非同期インタラプト・ライン

(1)  $\overline{\text{INT0}} - \overline{\text{INT7}}$ 

この 8 つのプライオリティ・インタラプト・リクエスト・ラインは、パラレルのインタラプト解決回路で用いられる。 $\overline{\text{INT7}}$  が最も低く、 $\overline{\text{INT0}}$  が最も高い優先権を持つ。

(2)  $\overline{\text{INTA}}$ 

$\overline{\text{INTA}}$  は、外部ロジックが Multibus データ・ラインにインタラプト・ベクタの情報を設定することを要求するために、バス・マスタによって用いられる。

## 9.7 パワー供給ライン

各種の安定化されたパワーを供給するラインがバスに備わっている。各モジュールは、大容量コンデンサと、存在するロジック素子に局所的な高周波用コンデンサとを共に備える必要がある。

## 9.8 予備ライン

予備ラインは用いてはならず、将来における Intel の定義に利用可能な状態にして置く必要がある。

Multibus は論理的に、8086 の分離バスと類似している。アドレスは、アドレス・ライン  $\overline{\text{ADR0}}$  から  $\overline{\text{ADR13}}$  で供給される（なお、アドレス・ラインのナンバーは 16 進数である）。

Multibusは、リード・コントロール信号の選択された素子への伝達が可能となる前に有効アドレスからの50nsのディレイを必要とする。リード・コントロール・パルスは少なくとも100nsの幅が必要であり、アドレスはリード・コントロール信号が終了して少なくとも50ns後も安定でなければならない。選択された素子が、100nsのリード信号あるいは指定されたアドレス・アクセス・タイムの最小値150ns以上を要するならば、素子は $\overline{\text{XACK}}$ （トランスファ・アクノリッジ）信号を用いてリード・サイクルを拡張できる。この信号は、8284 RDY 入力に接続されるレディ信号に同等である。 $\overline{\text{XACK}}$ は通常“ノット・レディ”で、素子がデータの受信あるいは送信に対してレディであり、バス・サイクルの終了が可能であることをCPUに通知するために、アクティブに駆動される。異なるタイプのCPUが混合したマルチプルプロセッサ・システムにおいて選択された素子の独立な動作を可能とするために、Multibusでは、読み込みまたは書き込みのコントロール信号に対してではなく $\overline{\text{XACK}}$ 信号に関してデータのセットアップとホールドのタイムを指定している。

ライト・バス・サイクルはリード・バス・サイクルに類似している。書き込まれるデータは、ライト・コントロール信号より最小50ns前に有効でなければならず、ライト・コントロール信号に続いて最小50nsは有効に維持されなければならない。

Multibusに付属のマスタ・モジュールは、最小のセットアップとホールドのタイムあるいはコントロール・パルス幅に違反してはならない。多くの設計では、最大帯域幅で動作させたときに最小の余裕よりも良い値が得られる。スレーブ・モジュールは、最小のセットアップとホールドのタイムを認められる必要があるが、適当な時間だけ $\overline{\text{XACK}}$ を返すのが遅ければ、アクセス・タイムを拡張する。

Multibusは、2つの基本的なインタラプト処理の方法を備えている。それは次のものである。

1. インタラプト・ベクタがバスで転送されない方法。インタラプト・ベクタはバス・マスタのインタラプト・コントローラによって生成される。割り込みを要求するスレーブは、バス・マスタとして同じモジュールの一部でなければならない。要求元スレーブが他のモジュールの一部ならば、スレーブは割り込みの要求のためにMultibusのインタラプト・リクエスト・ライン（ $\overline{\text{INT0}} - \overline{\text{INT7}}$ ）を用い、この割り込みはバス・マスタのインタラプト・コントローラによって処理される。
2. インタラプト・ベクタがバスで転送される方法。スレーブ素子が割り込みを要求すると、インタラプト・コントロール・ロジックはプロセッサに割り込みをかける。プロセッサは、 $\overline{\text{INTA}}$ ラインをローにしてシステム・バスをロックして割り込みを受け付ける。これによりインタラプト・ベクタの転送を可能にする。最初の $\overline{\text{INTA}}$ サイクルに続いて、インタラプト・コントロール・ロジックは、現在割り込みを要求している最も高い優先権のスレーブのアドレスを決定する。このアドレスは、アドレス・バス上に設定される。アドレス指定されたスレーブは、インタラプト・ベクタ・アドレスをマスタに伝送して応答する。

マスタとスレーブのモジュール設計に、標準の非同期データ伝送プロトコルとタイミングの仕様を与えてさらに、Multibusは複数のマスタがバス・コントロールの交換に用いる

標準的プロトコルを与える。非同期のマスタのバス共有を可能にするために、Multibusは、それに接続されるモジュールに局所的なクロック信号に独立なそれ自身のクロックを維持している。 $\overline{\text{BCLK}}$ で表わされるMultibusクロックは、バス・アクセスに対する非同期の要求の同期をとる。これにより、判定ロジックの優先権解決と一度に1つのマスタのアクセスを許可することが可能となる。

このバス判定法で、種々の速度で動作するマスタがシステム・バスの利用に対して公平に競うことができる。しかしマスタがシステム・バスのコントロールを獲得すると、転送速度は、Multibusクロック信号 $\overline{\text{BCLK}}$ ではなく、マスタの能力とそれに関連したスレーブ・モジュールだけに依存する。最小のアドレスのセットアップとホールドのタイムそして最小のコントロール・パルス幅を考慮すると、最大のバス帯域幅は5MHzである。バス判定と一般的なメモリ応答時間の付加的なオーバヘッドを考えると、実際の転送速度はしばしば2MHzのオーダになる。

Multibusは、バス・マスタ間でのシリアルまたはパラレルの優先権判定を可能にする。シリアル・プライオリティの方法を図9-1に示す。

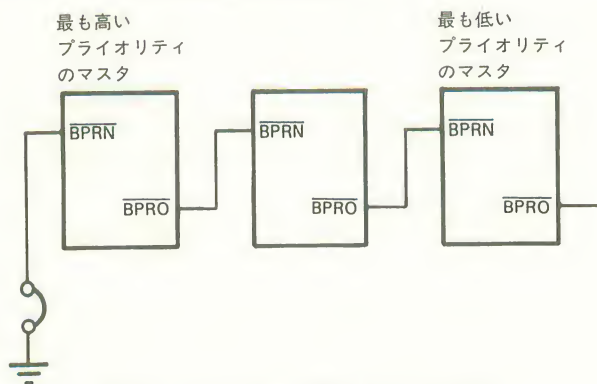


図9-1 シリアル・プライオリティの方法

最も高いプライオリティのマスタは $\overline{\text{BPRN}}$ をグラウンドに接続する。各マスタのプライオリティ・イネーブル出力は、次に低いプライオリティのマスタのプライオリティ入力 $\overline{\text{BPRN}}$ に接続される。そのマスタがバスを必要としなければ、 $\overline{\text{BPRN}}$ を $\overline{\text{BPRO}}$ に伝える。これは上位のプライオリティを下位のマスタに伝える。バスを必要とするマスタは $\overline{\text{BPRO}}$ にハイを出力する。これは下位レベルのマスタに対するプライオリティを否定する。このロジックは、システムにおける最も高いプライオリティのマスタからのプライオリティ・イネーブルを最も低いものへとディジィ・チェーンとして伝える。より高いプライオリティのマスタは現在のバス・マスタがすでに進行中のバス・サイクルを終了するまで待たなければならないので、他の信号はMultibusのアイドルまたは“ノット・ビジィ”の状態を示すために含まれる必要がある。この機能を果たす $\overline{\text{BUSY}}$ ラインは、すべてのバス・マスタに共通の信号である。Multibusを用いているマスタは、バスへのアクセスを得るためのその能力を決めるために、 $\overline{\text{BUSY}}$ の時間を記録している。



$\overline{\text{BCLK}}$ のハイからローへの遷移におけるサンプルで、 $\overline{\text{BPRN}}$ がロー（アクティブ）で、 $\overline{\text{BUSY}}$ がハイ（インアクティブ）であり、アイドル・バスを表わしているならば、現在のマスタがその伝送を終了する前により高いプライオリティのマスタがバスのコントロールを獲得するのを防ぐために、現在のマスタは $\overline{\text{BCLK}}$ の次のハイからローへの遷移の前に $\overline{\text{BUSY}}$ をローに駆動しなければならない。

現在のマスタがプライオリティを失なうと、伝送終了後に $\overline{\text{BUSY}}$ を解放してMultibusに対する接続をフロートにしなければならない。

他のマスタがバスの要求を必要とするならば、 $\overline{\text{BCLK}}$ のハイからローへの遷移の前に低いプライオリティのマスタをディスエーブルにしなければならない。さもないと、プライオリティ解決回路とより低いプライオリティのマスタにおいて競走条件が生じる。また、より高いプライオリティのマスタからのプライオリティ・ディスエーブルはディジィ・チェーンを通してより低いプライオリティのマスタに伝えられる必要があるので、最高から最低のプライオリティのマスタへの全体の伝播ディレイは1つの $\overline{\text{BCLK}}$ クロック期間を越えてはならない。これは、シリアルなプライオリティ判定が適応可能なマスタの数に上限を設定する。

パレルのプライオリティ判定は、 $\overline{\text{BCLK}}$ のハイからローへの遷移で各マスタがバス・

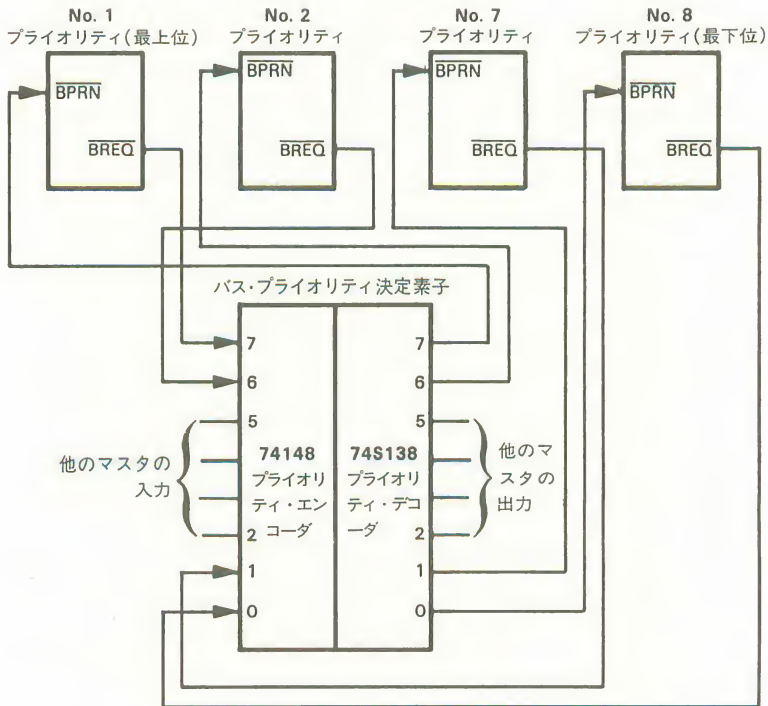


図9-2 パレルのプライオリティ決定とバス交換タイミング



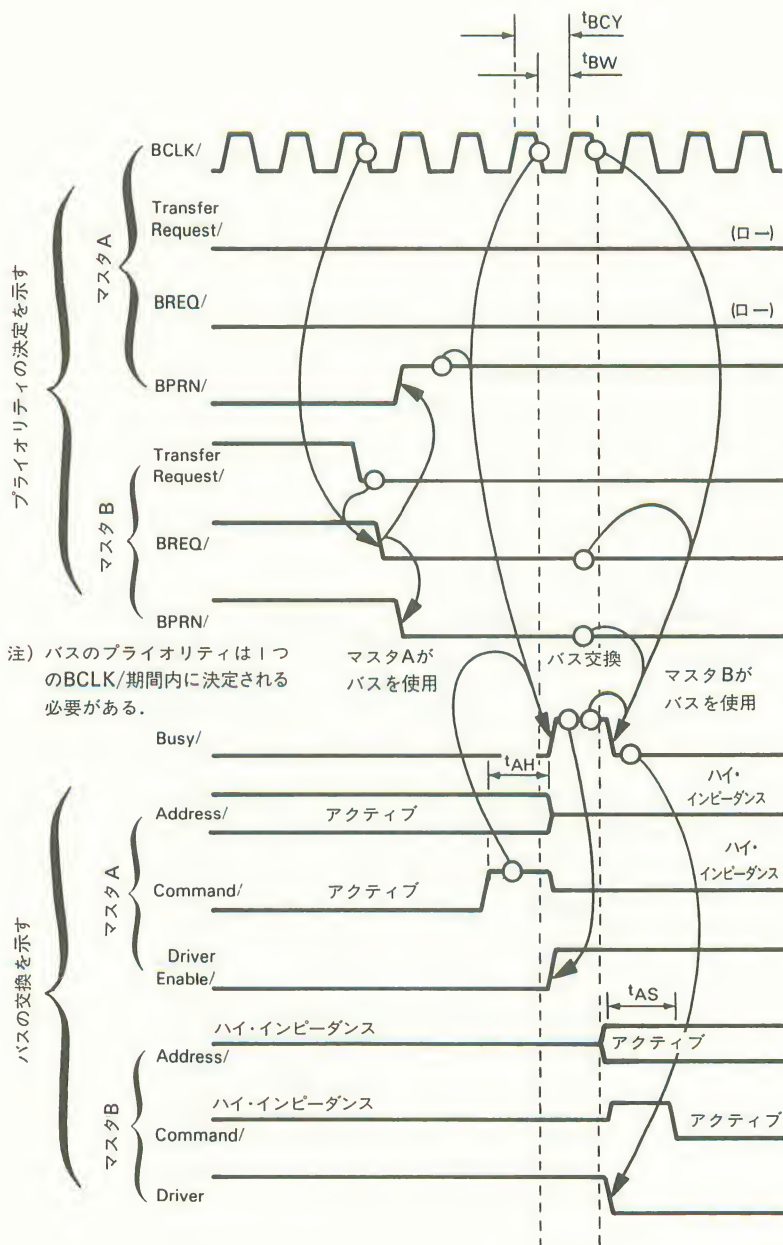


図 9-2 パラレルのプライオリティ決定とバス交換のタイミング(続き)

リクエスト  $\overline{\text{BREQ}}$  を発行し、外部のユーザ定義による回路がプライオリティを解決することを除けば、シリアルなプライオリティ判定に類似している。プライオリティの決定回路は、 $\overline{\text{BCLK}}$  の1期間内に各マスタに対して解決を行ない、安定なプライオリティの入力、 $\overline{\text{BPRN}}$  を戻さなければならない、パラレルの決定回路とバス交換のタイミングの例を図9-2に示す。

## 9.9 Multibus 構成の概念

Multibus 構成は、マルチプロセッサ・システムに対して定義の明確なバス構造を与える。バスは、マルチプロセッサ間の資源の共有とシステムのプロセッサ間の通信の手段として役立つ。マルチプロセッサ・システムには次の2つの基本的な形式が存在する。

1. 密結合システム。密結合のシステムでは、複数のプロセッサは共通のメモリ領域を通して互いに情報を交換することによって通信を行なう。
2. 疎結合システム。疎結合のシステムでは、複数のプロセッサはI/O構造を通して互いに情報を交換することによって通信を行なう。一般に、シリアルなコミュニケーション・リンクが用いられる。しかし、ある場合には情報は高速ディスクなどのマス記憶素子を通して伝えられる。

Multibus は密結合システムの必要条件を満たすように設計されているにもかかわらず、マルチCPU処理システムがBisyncまたはHDLCに似た標準I/Oコミュニケーション・プロトコルによって疎結合となることが可能である。バス共有の機構は、Multibusプロトコルとユーザ定義のプライオリティ決定回路の組合せから成る。Multibusは、バスのリクエストのために、プロセッサがバスのプライオリティを持つときに確認を受け取るために、そしてバスの有効性を示すために、各プロセッサに対して基本的なコントロールを与える。

ユーザはタスクに最も適切なプライオリティ決定の方法を選択する必要がある。プライオリティ・システムの選択には慎重でなければならない。さもなければ、Multibusはシステムの障害となり、システム全体の性能を悪化させることになる。バスの公平な利用を確実にするために、Multibusは共通のバス・リクエスト  $\overline{\text{CBRQ}}$  をサポートしている。この信号により、低いプライオリティの素子がより高いプライオリティのバス・マスタからバスを要求すること、あるいはより低いプライオリティのリクエストの保留を示すことが可能になる。これは、より高いプライオリティのマスタが強制的に（プライオリティを失なわせて）現在のマスタをバスから切り離すか、より低いプライオリティのマスタが（共通バス・リクエストによって）バスを要求するまで、現在のバス・マスタがバスのコントロールを維持することを可能にする。

共通バス・リクエストに対する現在のマスタの応答は、ユーザが定義できる。これは、マスタに現在の伝送の終わりかあるいはバス動作がなければ直ちにバスを解放させるか、またはマスタは単にリクエストを無視してより高いプライオリティのマスタに対してバスのコントロールを放棄するだけである。共通バス・リクエストは設計者に、システム・バスの利用を定義するとき別レベルのコントロールを与える。結果的に、現在のマスタ

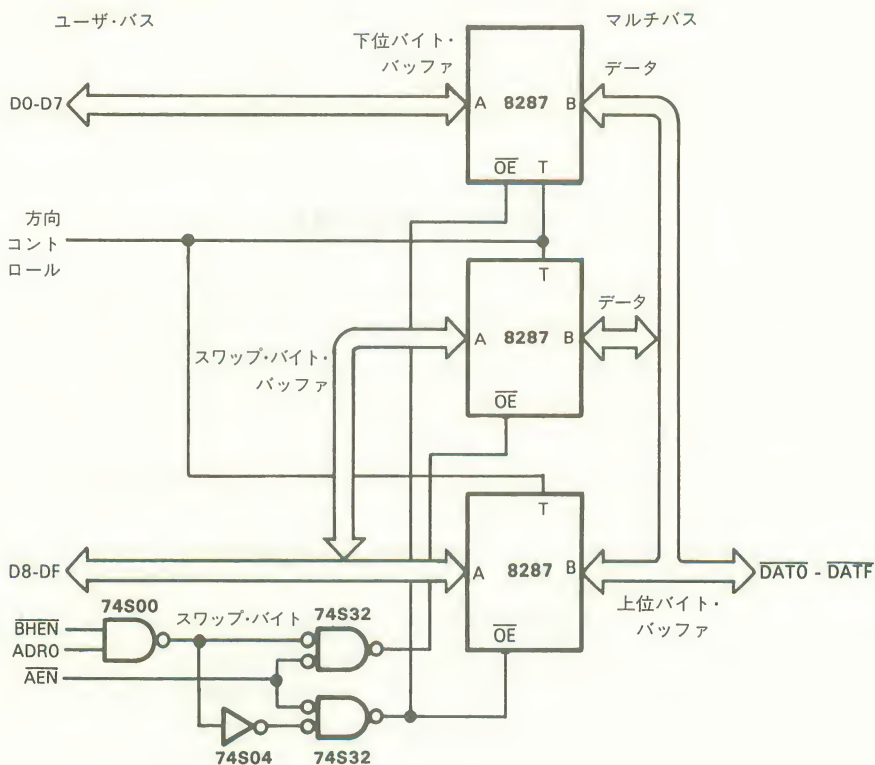


図9-3 8/16ビットの素子の伝送動作

は伝送のたびにバスの解放を強制されて再び要求することはない。これは、バスのアクセスと伝送に関するオーバーヘッドを最小にする。

Multibusの $\overline{\text{BUSY}}$ 信号により、より高いプライオリティのマスタが強制的にプライオリティを失なわせようとしても（強制的にその $\overline{\text{BPRN}}$ 信号をインアクティブにする）、現在のマスタはバスのコントロールを維持できる。これは、現在のマスタのバス・サイクルが他のバス・マスタに対して実行されるバス・サイクルで分離できない、複数のバス・サイクルを要する動作に必要である。現在のバス・マスタがバスのコントロールを有していれば、バスが利用できることを示す $\overline{\text{BUSY}}$ を解放せずに、バスのコントロールを続ける。この機能は、複数バス・サイクルのインタラプト・アクノリッジ・シーケンスとテスト・アンド・セットの動作に必要とされる。バスのコントロールを放棄してはならないこれらの条件を解釈するのはマスタ・モジュールの機能である。この条件が起きるときは、マスタは必要な動作の終了まで $\overline{\text{BPRN}}$ を無視しなければならない。

Multibusは、アドレスとデータのラインが別々の分離システム・バスである。システムは20のアドレス・ラインで1メガバイトのアドレス領域を維持し、16ビットのデータ・バ

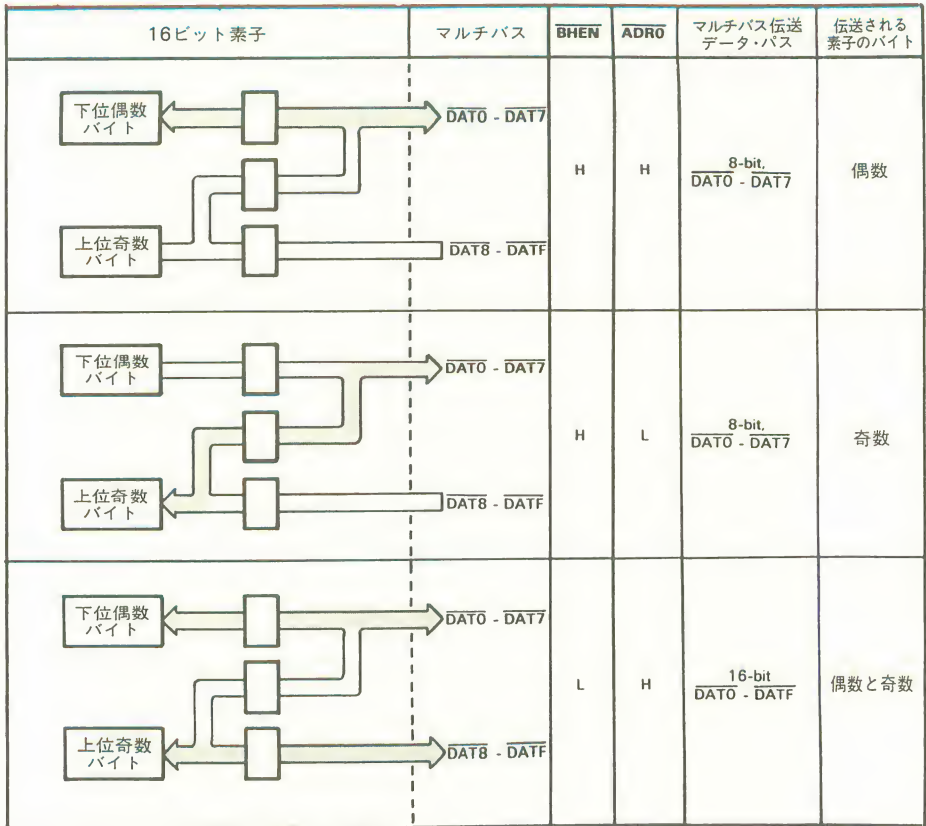


図9-3 8/16ビットの素子の伝送動作(続き)

スが存在する。8ビットのデータ・バスを維持するだけのMultibusとコンパチブルのモジュールとの互換性から、すべての8ビット伝送はアドレスを無視してデータ・バスの下位を通して行なわれる。16ビット伝送だけが16のデータ・ラインすべてを用いる。これは、データ・バスの上位と下位で上位と下位のバイトを送送する標準的な8086の方法とは別のものである。結果的に、既存の8ビット・インターフェイスのスレーブ・モジュールに対するバイト伝送はCPUと独立している。

16ビット・インターフェイスをサポートするスレーブ・モジュールは、8ビット・データ・バスにだけインターフェイス可能なマスタ・モジュールによって一度に1バイトのアクセスができ、さらにスレーブは、16ビット・データ・バスすべてをサポートするマスタによって一度に1ワードあるいは1バイトのアクセスができる。Multibusの信号BHENは、バイトあるいはワードのどちらの伝送が行なわれているかを判断するために必要な付加的情報を与える。図9-3に、Multibusのシステム・バスとの間でバイトとワードの情報の切り替えをBHENがどのようにコントロールしているかを示す。





# 第10章

## 8086のマルチプロセッサ構成

8086ファミリーの構成要素は、シングル・プロセッサとマルチプロセッサの両構成を可能とするシステム・アーキテクチャを与えるように考案されている。この章では、マルチプロセッサ・システムにおけるマキシマム・モードの使用法について述べると共に、8086ファミリーの構成要素でサポートされる各種の構成を検討する。

8086ファミリーのマルチプロセッシングの機能は、次の2つの別々の異なる特徴に基づいている。

1. ローカル・バス上に位置し、8086CPUの基本的アーキテクチャを強調する、特殊機能のプロセッサ。
2. 共通のシステム・バスを共有するマルチプルCPU。

特殊機能のプロセッサは1つのCPUに寄与し、CPUの命令セットの拡張を行ない、このCPUと並列に処理する。したがって、特殊機能のプロセッサはコープロセッサと呼ばれる。対照的に、マルチプル・マイクロプロセッサ・システムは、CPUの繰り返しユーザで定義される古典的なマルチCPUの環境と酷似している。システムでは2つの機能が組合される、すなわち、マルチプロセッサ・システムにおいて各CPUはそれ自身に寄与するコープロセッサを有することに注意。マルチCPUのマルチプロセッサ・システムの要求は、9章で定義されているMultibusによって満たされる。

### 10.1 コープロセッサ

8086は、コープロセッサの利用を可能とすることを目的としたハードウェアとソフトウェアの特徴を有している。ハードウェアのサポートには、キュー・ステータスの信号(QS0, QS1), TEST入力, そしてCPUのローカル・バスを共有するための機構( $\overline{RQ}/\overline{GT}$ )が含まれる。ソフトウェアのサポートは、コープロセッサを動作させるESCAPE命令と呼ばれる特殊なクラスの命令と(CPUとコープロセッサのソフトウェアの

同期に用いる)  $\overline{\text{TEST}}$  入力をサンプルする  $\text{WAIT}$  命令より成る。

コープロセッサは CPU のローカル・バスと直接にインターフェイスする。CPU のローカル・バスには、多重化されたアドレス/ステータスとアドレス/データのライン、 $\overline{\text{S0}}$ 、 $\overline{\text{S1}}$ 、そして  $\overline{\text{S2}}$  のステータス・ライン、 $\overline{\text{QS0}}$  と  $\overline{\text{QS1}}$  のキュー・ステータス・ライン、 $\overline{\text{TEST}}$ 、 $\overline{\text{READY}}$ 、 $\overline{\text{RESET}}$ 、 $\overline{\text{RQ}}/\overline{\text{GT}}$  ラインの 1 つ、そしておそらく  $\overline{\text{LOCK}}$  のラインが含まれる。 $\overline{\text{LOCK}}$  信号に対する要求はコープロセッサに依存して変わる。コープロセッサは次の 2 つの理由から、ローカル・バス上に位置することが許されている。

1. コープロセッサは CPU の動作をモニタすることができる。
2. コープロセッサは CPU の資源に対して完全なアクセスを有する。

コープロセッサは、 $\overline{\text{RQ}}/\overline{\text{GT}}$  ラインの 1 つを通してローカル・バスのコントロールを要求する。CPU がバスのコントロールをコープロセッサに譲った後に、コープロセッサは CPU とまったく同じにローカル・バス・サイクルで動作する。コープロセッサがローカル・バスの使用を終われば、バスのコントロールを得るために用いられたのと同じ  $\overline{\text{RQ}}/\overline{\text{GT}}$  ラインを通して CPU にコントロールが返される。このインターフェイスの例を図 10-1 に示す。

コープロセッサは以下のシーケンスで動作状態になる。CPU が命令のフェッチを行ない命令を実行している間、コープロセッサは  $\text{ESCAPE}$  命令を待ち受けて命令の流れをモニタしている。メモリから CPU の命令キューに伝送される情報は (命令でオペランドとして用いられるメモリ・データとは対照的に)、命令フェッチを識別するステータス・ライン ( $\overline{\text{S0}}$ 、 $\overline{\text{S1}}$ 、 $\overline{\text{S2}}$ ) をデコードすることによって選択される。 $\overline{\text{A0}}$  と  $\overline{\text{BHE}}$  は、シングルまたはダブルのどちらのバイトの命令フェッチが実行されるかを判断するためにデコードされる。

キューより得られる情報は、キュー・ステータス・ライン ( $\overline{\text{QS0}}$ 、 $\overline{\text{QS1}}$ ) によって、命令の第 1 バイトあるいは後続バイトとして識別される。キュー・ステータス・ラインはまた、バイトのフェッチが行なわれないこと、あるいはキューが空であることを示す。コープロセッサのキュー (これは CPU のキューに追従している) からの情報が  $\text{ESCAPE}$  命令を示し、キュー・ステータスがそれを命令のオブジェクト・コードの第 1 バイトとして識別するならば、コープロセッサは動作を行なう。 $\text{ESCAPE}$  命令のオブジェクト・コードを次に示す。



X は CPU に対するドント・ケアのビットであるが、コープロセッサに対して 64 の可能な命令を表わすことができる。 $\text{ESCAPE}$  命令に応じて、CPU は通常のアドレッシング・モード指定として mod と r/m のフィールドを用い、指定アドレスからデータを読み込むためにバス・サイクルを実行する。コープロセッサは、選択されたメモリ位置のア

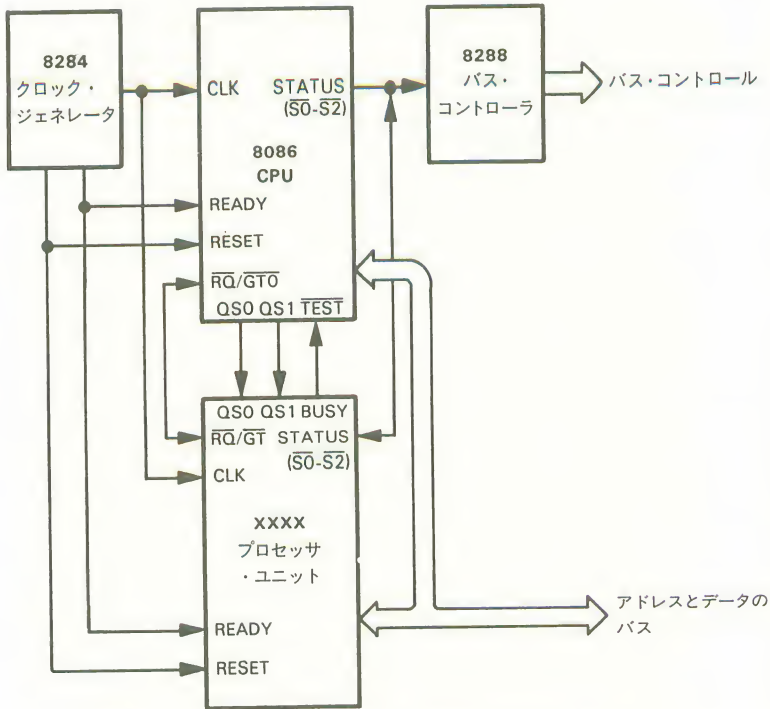


図10-1 マキシマム・モード・マルチプル・プロセッサ

アドレスとデータの両方を捕えることができる。この機構は、プログラマがメモリ・アドレッシング・モードのすべての範囲でコープロセッサに対して定義されたESCAPE命令を通常のCPU命令として取り扱うことを可能にする。1つのパラメータと共にコープロセッサにアドレスを受け渡すことができる。コープロセッサがメモリとの間で付加的データを転送する必要があるば、CPUのバスのコントロールを要求する。コープロセッサはCPUのレジスタに対するアクセスを持たないので、コープロセッサで用いられるすべての情報はメモリ内に存在しなければならないことに注意。

CPUは、ESCAPE命令の間に読み込まれるデータを捕えず、リード・バス・サイクルを除いてこの命令を無効操作として取り扱う。ESCAPE命令の実行後、CPUとコープロセッサは自由に並行して特定のタスクの実行を続ける。命令実行の間、コープロセッサは一般に動作中であることを示すためにTESTのラインをハイに保つ。プログラムでコープロセッサが動作中でないことを保証できないならば、別のESCAPE命令を実行する前に、プログラムで（可能ならば）コープロセッサのステータスを読み出すか各ESCAPE命令の前にWAIT命令を挿入しなければならない。

WAIT命令は、ESCAPE命令を実行する前にコープロセッサが動作中ではなくなるまで強制的にCPUをウェートさせる。WAIT命令の間、多分、命令キューを満たすことを除いて、CPUはバスを必要としない。しかし、CPUはWAITサイクルの間（イ

ネーブ状態ならば) 割り込みに応答する。E S C A P E 命令実行中にコープロセッサはCPUの命令の流れをモニタし続けていることに注意。インテルは8087コープロセッサの発表だけを行なっているが、インターフェイスの汎用性から、数値、言語のサポートなどに対する広い範囲のより高いレベルのコマンドに適用可能である。

## 10.2 共有システム・バスにおける多重処理

マイクロプロセッサのアプリケーションがより複雑となり、マイクロプロセッサのコストとパフォーマンスの比が小さくなるに従い、1つ以上のマイクロプロセッサでシステムを設計することがコスト的により有効になっている。マルチプロセッサ・システムは、デザインを容易に識別可能で他のシステムに対する明確な通信インターフェイスを有する機能的なサブシステムに分割することによって実現される。デザインを分割してハードウェア・インターフェイスとソフトウェア・インターフェイスを定義した後に、全体の最終結果設計時間を減少させる手段として個々のチームによって並行して、各サブシステムの設計と開発が行なわれる。この方法はすべての設計チームの間に高度な協力と調整を必要とするが、モジュラリティ、拡張性、そしてメンテナンスの容易さなどの利点がある。

システムをマルチプロセッサ分散インテリジェンス・システムに分割するとき、機能的サブシステムの必要条件が、疎結合、密結合、あるいは疎結合と密結合のプロセッサの必要性を示す。

以下の議論では、疎結合プロセッサは共有の I/O 機構を通して通信を行ない、密結合プロセッサは共有メモリを通して通信を行なう。疎結合のプロセッサに対して、各CPUは、それに付属の他の I/O 素子として取り扱われる I/O インターフェイスを通して、他のCPUと通信を行なう。このタイプのインターフェイスには、SDLC, ASYNC, GPIBなどの広範な種類の標準的プロトコルが存在する。これらのインターフェイスは、サブシステムのハードウェアが I/O インターフェイスを備えて、さらにソフトウェアがインターフェイスをコントロールして、メッセージ・プロトコルを解釈することを要する。

疎結合サブシステムは一般に、頻度の高いあるいは高速の通信を必要とせず、お互いに数マイルすら物理的に離れている。

マルチプル・プロセッサが密結合であれば、プロセッサ間でメモリを共有するための方法が与えられなければならない。8086ファミリーは、共有システム・バスへの結合によってメモリを共有する。このバスによってアクセス可能なメモリと I/O の素子は、バス・マスタとなる資格のあるすべてのCPUによって利用される。

共有のメモリと I/O の素子は、希望するプロセッサ間通信機構を与える。

マルチマスタ・システム・バスに対する基本的CPUインターフェイスを図10-2に示す。

図10-3は8086システムの基本的インターフェイスを示す。8289バス・アービタは、バス・コントロールの移動に必要なプライオリティの判定とプロトコルのロジックを有する。このロジックはMultibusについて第9章に述べられている。8288バス・コントローラはCPUに対するシステム・バス・コントロール信号を備えており、システム・バスとのCPU



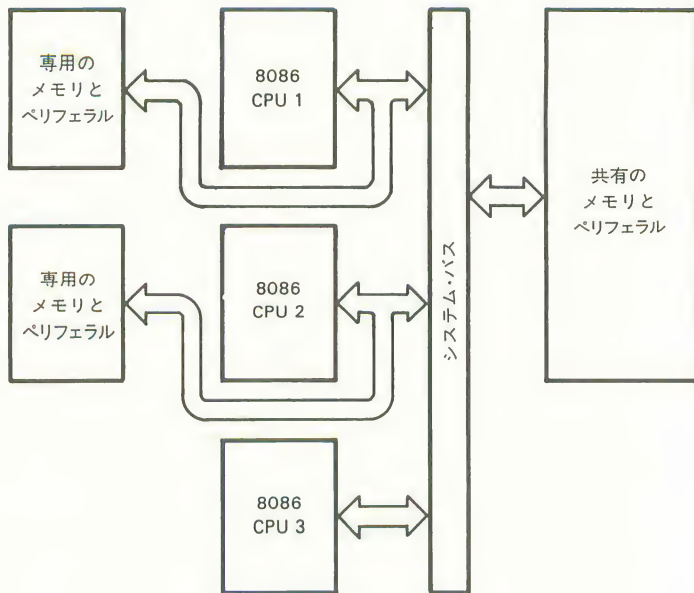


図10-2 マルチプロセッサ構成

のアドレスとデータのバス・インターフェイスも配慮している。8283と8287は共有システム・バスへのCPUのインターフェイスを実現する。8288と8289はまた、システム・バス上の共有資源またはCPUのローカル・バス上の固有資源への移動を管理する。

図10-3では、すべての資源、したがって、すべてのバス伝送はシステム・バスによって管理されている。バス・サイクル開始のために、8086は、パッシブ状態 ( $\overline{S2}, \overline{S1}, \overline{S0} = 1, 1, 1$ ) からアクティブ状態の1つにステータス・ラインを駆動する。8288と8289とCPUは共通クロックCLKによって同期的に動作し、ステータス・ラインをモニターすることによってバス・サイクル・リクエストを検出する。8288は、ローカルの多重化バスから8283へのアドレスのストロープのためにALEを発行する。8289がシステム・バスのコントロールを持たない ( $\overline{BUSBY}$ をローに駆動していない) ならば、コントロールを得るためにバス・リクエスト ( $\overline{BREQ}$ ) と共通バス・リクエスト ( $\overline{CBRQ}$ ) を発行する。このプロトコルはMultibusのシステム・バスについて述べたものと同一である。

8289は、バス・コントロールを得るまで  $\overline{AEN}$  をインアクティブ (ハイ) 状態に維持する。この動作は、8283のシステム・バスのアドレス駆動と、8288のバス・コマンドの駆動あるいは ( $\overline{DEN}$  で) データ・トランシーバをイネーブルすることを防止する。8283は、 $\overline{OE}$  入力の状態とは無関係に、ALEの間にアドレスを捕える。 $\overline{AEN}$  のインアクティブ状態はまた、システムからの8284RDY入力をディスエーブルとするのにも用いられる。RDY入力のディスエーブルにより、このインターフェイスがバスのコントロールを獲得してバス・サイクルを終了するまで (ウェイト状態の挿入によって) 強制的にCPUをウェイトさせる。一度8289がバスのプライオリティを獲得してバスが動作中でなければ、 $\overline{AE}$



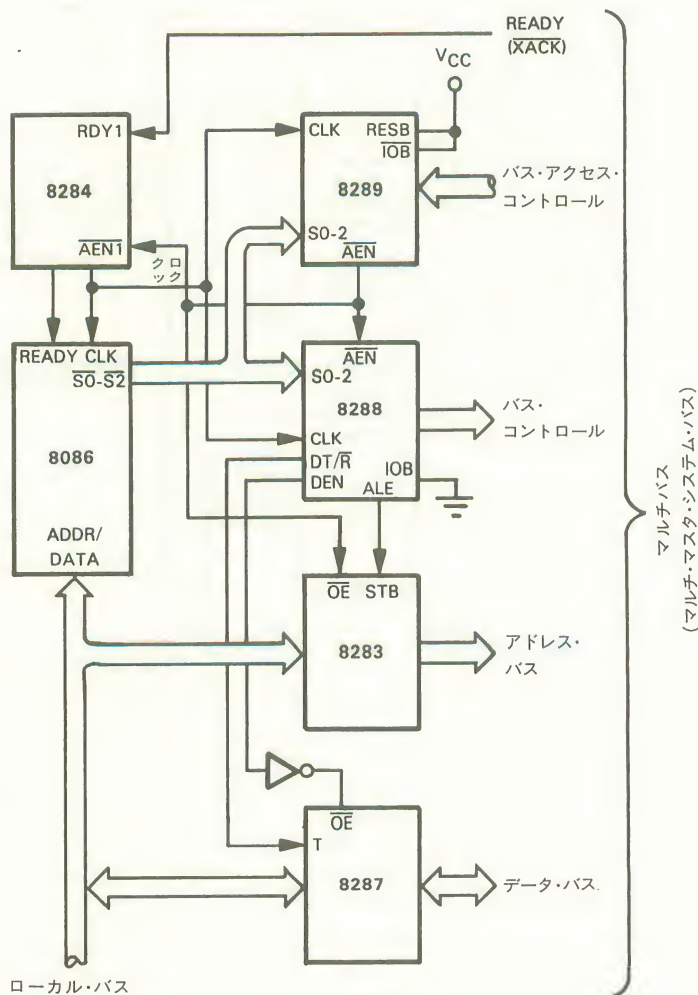


図10-3 ローカル資源のないCPU

$\overline{N}$ をイネーブルとして他のバス・マスタに対して $\overline{BUSY}$ を主張する。 $\overline{AEN}$ は直ちにバス上のアドレスをイネーブルにして、8288がデータ・バスのトランシーバ(8287)をイネーブルとすることを可能にする。

システムにおけるアドレス・セットアップとチップ・セレクト・デコードのタイムを可能とするために、105から275ns後まで8288はコマンドをイネーブルとしない。8284とCPUがAENのイネーブルとなった直後にレディを検出してバス・サイクルを早まって終結することを防止するために、システムからのRDY入力は通常インアクティブで、8288によってコマンドが発行されるまでディスエーブルでなければならない。これを満たすために、伝送に関係して選択された素子は一般に、伝送が終了するまでRDY (Multibusのシ

システム・バスの定義では  $\overline{\text{XACK}}$  を返さない。伝送を終了する時間は素子に依存しているので、各素子はコマンドの選択から  $\text{RDY}$  (または  $\overline{\text{XACK}}$ ) までに適当な遅延を備えている。

レディの検出後、CPU はステータス・ラインをパッシブ状態に戻す。この動作は、8288 がコマンドを落としてトランシーバをディスエーブルにすることによって行なうバス・サイクルの 8288 の決定を可能にする。より高いプライオリティのバス・マスタが (バス・プライオリティ  $\overline{\text{BPRN}}$  を失うことによって) バス・サイクルの間にこのバス・マスタを強制的にバスから切り離そうとするならば、8289 はステータス ( $\overline{\text{S2}}$ ,  $\overline{\text{S1}}$ ,  $\overline{\text{S0}}$ ) がパッシブ状態に戻るまで  $\overline{\text{BUSY}}$  をアクティブ (ロー) に保つことによってバス・コントロールを維持する。8289 がバスのコントロールを持てば、CPU によって開始されるバス・サイクルが直ちに実行されるのを可能にするために、 $\overline{\text{AEN}}$  をアクティブに維持する。このような状況でのタイミングと信号シーケンスは、シングル CPU のマキシマム・モード・システムと同一である。8289 によってサポートされるバス放棄条件の十分な補足はこの章の後で述べる。

以前の解説では、すべてのメモリと I/O がマルチマスタ・システム・バスに接続されているシングル CPU を表現している。バス・サイクルに利用可能なクロック・サイクル数を考慮すると、バス・サイクルにつき 4 クロックを仮定して、実行されるバス・サイクルの数を計算すると、応用が計算の限界かどうかには依存して、シングルの 8086 CPU は利用可能なバス帯域幅の 50% と 80% の間を利用できることが明らかになる。2 つの 8086 が共有バスに所属しており、両プロセッサに対するすべてのメモリと I/O がこの共有バスに接続されていれば、各 CPU についてのスループットは、次式から 37% にまで悪化する。

$$\frac{80-50}{80} = 37\%$$

この悪化は、共有バスへのアクセスに対する競合の直接の結果である。この計算では、各 CPU が利用可能な全体のバス・サイクルの 80% を利用するが、他の CPU と 1 つのバスを共有しているので、利用可能なバス・サイクルの 50% だけを与えられると仮定している。より多くの CPU がシステムに付加されれば、それだけすべての CPU に悪化が起きる。マルチプロセッサ・システムから評価しうる利益を得るためには、高度な同時処理が存在しなければならない。したがって、共有バスへのアクセスの競合によるディレイは最小でなければならない。

この問題に対する 1 つの解として、8086 ファミリーは各 CPU に対して固有と共有の資源が定義されるのを可能にしている。固有の資源はメモリや I/O を含み、これは CPU の固有またはレジデント・バスに接続され、割り当てられた CPU によってだけアクセス可能である。それと対照に、マルチマスタ・システム・バスに直接に接続され、1 つ以上のマスタによってアクセス可能なメモリと I/O から共有資源は構成される。

資源分割の主要な方法、アドレス・マッピングとメモリ対 I/O は、8288 と 8289 によってサポートされる。図 10-4(a) から 10-4(c) までは、この方法から導かれる変形を示す。これらすべての例における共通の要素に、第 2 の組のアドレス・ラッチを含むことと 8284 の 2 つ

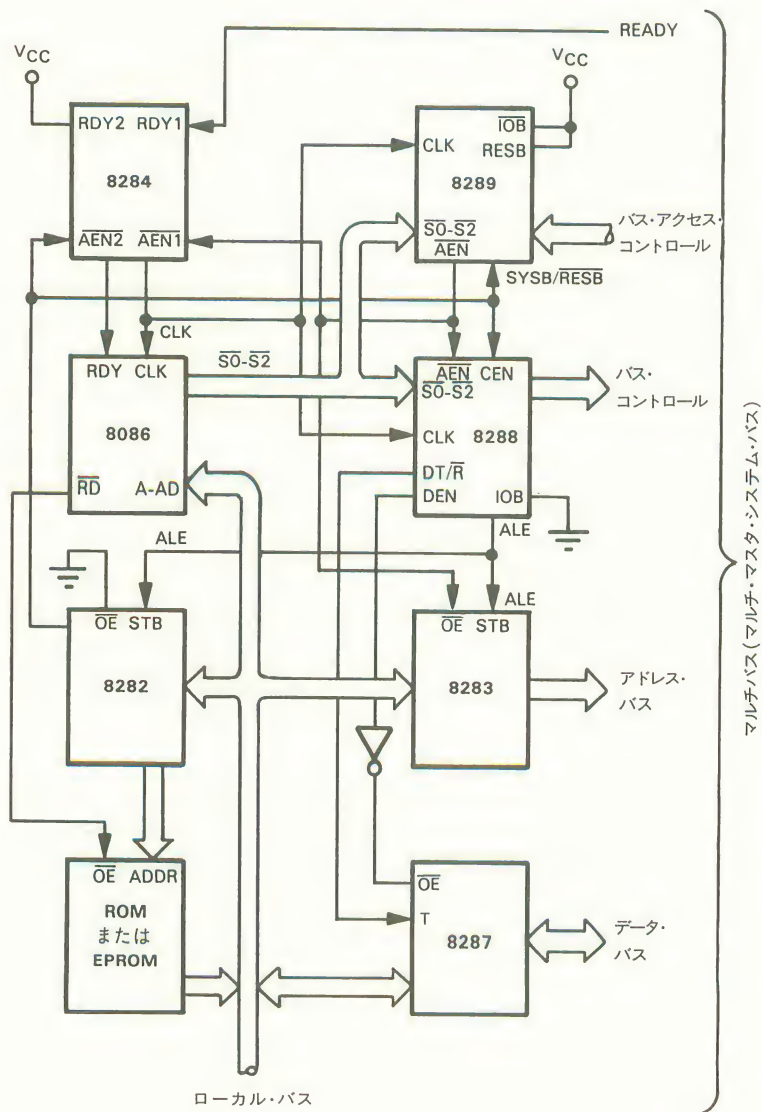


図10-4(a) ローカルROM / EPROMを有する8086



図10-4(c) ローカルRAM/ROM/EPROM/I/Oを有する8086



のRDY入力の使用がある。図10-4(a)から10-4(c)に示す構成の主要な差異は、固有資源のサポートに必要な付加的要素の数とサポートされる固有資源のタイプにある。

図10-4(a)はCPUのローカル・バス上にROMあるいはEPROMの存在を認め、付加的アドレス・ラッチと可能性のあるアドレス・デコード・ラッチだけを必要とする。8289はレジデント・バス・モードに切り替えられ(RESBをVccに)、これは8289にこのCPUがレジデント・メモリをサポートすることを通知する。8288の切り替えによる選択は、図10-3に示す構成と異ならない。バス・サイクルの間に、8288はALEを発行し、8283と8282にアドレスをラッチする。レジデント・メモリに割り当てられたアドレス領域はデコードされて、このバス・サイクルがレジデントあるいはシステムの資源を用いるかの判断に利用される。バス・サイクルがシステム・バスを利用するならば、デコードの結果は、8289に対してSYSBを、8288に対してCENを示し、関連のあるAENによってローカル・バスのRDYをディスエーブルにするために、ハイでなければならない。

8289がRESBモードのとき、8289に対するSYSB/RESB入力は、8289がシステム・バスを要求するか解放するかを決定する。SYSB/RESBがハイならば、8289はシステム・バスを要求し、SYSB/RESBがローならば、8289はバスを解放する。8288のCEN入力は、コントロール出力がレジデント・バス・アクセスの間にアクティブ状態に駆動されるのを防止する。この信号は、8289がバスのコントロールを有するならば、コントロール信号がシステムに対して発行されるのを防ぐ。8289がバスのコントロールを有する間は、8288はコントロール出力をアクティブ状態に維持しなければならないので、入力によって8288はコントロール出力をフロートにはしない。8288は常にイネーブル(OEはローに接続)になっていることに注意。これにより、各バス・サイクルのアドレスの適当なシステム・バスまたはレジデント・バスの選択への早いデコードが可能となる。

8288にラッチされる安定なアドレスはまた、固有のROMまたはEPROMにチップ・セレクトとアドレスを供給する。8086からのリード信号(RD)は、選択された固有素子の固有多重化バスへのデータの駆動をイネーブルとするために用いられる。固有資源が選択されると、8284のAEN2入力は、ローカルのRDY2のCPUに対するレディの生成を可能とするために、イネーブルとなる。8288はシステム・バスのコントロール信号を駆動しないので、8289がバス・コントロールを有してAEN1がアクティブであっても、RDY1(またはXACK)を受け付けることはできない。ローカル資源がローカル多重化バスからのバッファを必要とするならば、ローカルのチップ・セレクトとRDはバッファをイネーブルとするために用いられる必要がある。I/Oアドレス領域がローカル・メモリ・アドレス領域とオーバラップするならば、I/Oアドレスがレジデント・バス・アクセスにデコードされるのを防止するために、S2はラッチされてローカル・アドレス・デコードに含まなければならない。

I/Oコマンドまたはライト・コマンドはローカル・バスに利用できないので、この方法はメモリ領域のローカルなROMとEPROMに対してだけ動作する。ただし、オブジェクト・コードのアクセスは一般にバス・サイクルの50%から70%を利用することに注意。したがって、CPUによって実行されるオブジェクト・コードの大部分がローカルのRO

MあるいはEPROMに存在すれば、CPUあたりのシステム・バス利用度は30%以下にまで低下する。すべてのRAMとI/Oは共有システム・バスに位置せねばならず、したがって競合の可能性を持つ。

図10-4(b)は、図10-4(a)に示す前の構成にローカルのI/Oを付加している。これは、8288と8289と共にIOB(I/Oバス)モードの動作に切り替えることによって達せられる。このモードでは、すべてのI/Oはローカル・バス上に位置すると仮定され、したがってI/Oはシステム・バスに対するアクセスを要求しない。しかし、IOBの選択が用いられるとき、メモリ・マップでなければ、CPUはシステム・バス上のI/Oにアクセスすることができない。各I/Oのバス伝送に対して、8289はバスのコントロールを獲得しようとはせず、また現在コントロールを有してもバスを解放する。 $\overline{\text{AEN}}$ 入力の状態とは無関係に、8288はI/Oのコントロール出力を駆動する。このモードでは、8288のI/Oコントロール出力は、システム・バスのコントロール・ラインに接続されてはならない。これは、ローカルのI/O素子に対するコントロールを供給するためにだけ用いられる。図10-4(a)に対して、この拡張をサポートするために付加的な回路は必要としない。

8284の $\overline{\text{AEN}}2$ 入力は、ローカル・メモリあるいはローカルI/Oに対するチップ・セレクトによってイネーブルとなる。8288のCEN入力は、I/Oコントロール出力がロー(アクティブ)に駆動されるのを可能とするために、I/Oサイクルの間はハイでなければならない。I/O素子に対するデータ・ラインがローカル多重化バスとの間にバッファが必要ならば、8288からのDT/ $\overline{\text{R}}$ 信号はバッファに対する方向コントロール信号として用いることができる。バッファをイネーブルとするために、8288は別のペリフェラル・データ・イネーブル・コントロール(PDEN)を備えている。この信号は、8288がI/Oバス(IOB)モードに切り替えられているときにだけ利用できる。 $\overline{\text{PDEN}}$ はI/O伝送に対してイネーブルとなり、DENはシステム・バスによるデータ伝送に対してだけイネーブルとなる。

別の構成に対する次の拡張を図10-4(c)に示す。このケースに対しては、第2の8288とデータ・バス・トランシーバが付加される。この構成は、完全にアドレス・マッピングに基づいているので、最も融通性があり、共有とローカルのROM/EPROM、RAMとI/Oへのアクセスが可能である。IOBモードはどちらの8288にも用いられていない。ローカル・バスはマルチマスタ・バスではないので、第2の8289は必要とされない。アドレス・マップにおいてPROMあるいはデコード・ソースがメモリとI/Oの参照に対して存在する。結果のSYSB/ $\overline{\text{RESB}}$ 信号は、レジデント・バスの8288CEN入力に対して適当なイネーブルの極性を供給するために反転される。8288は、レジデント・データ・バスに対してアドレスのラッチと完全なデータ・バスのトランシーバ・コントロールを、そしてメモリとI/Oに対して読み込みと書き込みのコントロールの完全な集合を供給する。マルチマスタ・システム・バス・インターフェイスは、図10-4(a)の構成に対して述べたのと同様に機能する。

図10-4(c)の構成は、固定されたプログラムと定数のためのレジデントROM/EPROM、スタックとローカル変数のためのレジデントRAM、そしてこのCPUだけがアクセ

スを必要とするレジデント I/O を、CPU がサポートすることを可能にする。結果は、共有のメモリまたは I/O によってコントロールとデータを伝えるモジュールの完備した処理モジュールとなる。これにより、CPU あたりの代表的なシステム・バスの使用が 25% 以下にまで減少したシステムで、高度な同時処理が可能となる。

CPU に対する多重バスの概念の最後の拡張は、レジデント・バスではなくて第 2 のマルチマスタ・バスの実現である。これはフォールト・トレラント・システムに有用であり、また、複数プロセッサ間コミュニケーション・チャンネルに基づく性能の増強を可能にする。この場合、システム・バスにアクセスしようとする CPU は、プライマリ・バスを利用しようとするが、これが利用できないならば、セカンダリ・バスにアクセスする。

### 10.3 8289のバス・アクセスとリリース・オプション

RESB と IOB のモードに加えて、8289 はマルチマスタ・システム・バスの利用を最適にするいくつかの他のオプションを有している。この付加的機能は、バスの要求よりも解放に影響を与える。バス・リクエストの動作に関して、1 つのコメントを加える必要がある。RESB モードを除いて、T2 のローからハイへの CPU クロックの遷移に続く第 2 のハイからローへのクロック遷移 (BCLK) で、8289 はバス・リクエストを発行する。RESB モードに対する 1 つの CPU クロックのディレイは、安定な SYSB / RESB 信号発生のためのアドレス・デコードの時間を与える。

バス・コントロールの解放についての付加的コントロールのために、8289 は LOCK、ANYRQST、そして CRLCK の入力を備えている。一般に、8289 はプライオリティを失なった (BPRN がハイになった) ときに、現在のサイクルの終わりまたは (バス・サイクルが進行中でなければ) ただちにバスのコントロールを放棄する。LOCK により、バス・プライオリティとは無関係に、バス・コントロールを維持する (BUSY をローに駆動し続ける) ことができる。LOCK 先行の命令実行の間とインタラプト・アクノリッジ・シーケンスの間に CPU がバス・コントロールを維持することを保証するために、8289 の LOCK 入力は CPU の LOCK 出力によって駆動されなければならない。LOCK 信号により、ロックされた交換のような動作が、ノンリエントラント共有資源の基本的セマフォ・コントロール (クリティカル・コード・セクションとしても知られている) に対してプリミティブとして動作することが可能になる。

ANYRQST は、コモン・バス・リクエスト (CBRQ) がアクティブならば、現在のバス・サイクルの終わりあるいは (バス・サイクルが進行中でなければ) 直ちに強制的に 8289 にバスを解放させる、切り替えによる選択である。

ANYRQST の主張は、強制的により低いプライオリティのバス・マスタへ 8289 にバスを解放させる。このケースに対するバスの解放には、8289 がそのバス・リクエストを落とし (BREQ をハイに戻し)、BPRO (バス・プライオリティ・アウト) をイネーブルとする必要がある。(BREQ は並列プライオリティ解決回路に用いられ、一方 BPRO は次に低いプライオリティの BPRN に対する BPRO のディジィ・チェーンによる直列プラ



イオリティ判定方法をサポートするために供給されることに注意。2つの方法のうち1つだけが単一のシステムに適用される。

次いで8289は、 $\overline{\text{BUSY}}$ を解放することによって1つのバス・クロック期間後にバスを解放する。より低いプライオリティの8289が $\overline{\text{BPRN}}$ を受け取ることを可能とするために、8289は $\overline{\text{BREQ}}$ と $\overline{\text{BPRO}}$ を解放しなければならない。他の8289をバスから強制的に切り離すより高いプライオリティのマスタの通常のケースに対して、より高いプライオリティのマスタはプライオリティの機構から $\overline{\text{BPRN}}$ を受け取り、より低いプライオリティのマスタがその $\overline{\text{BPRO}}$ または $\overline{\text{BREQ}}$ を解放することを必要としない。

ANYRQSTがハイに切り替えられて $\overline{\text{CBRQ}}$ がローに接続されていれば、8289はバス・サイクルの終了後にバスを解放する。これはバス伝送ロジックに高度なオーバーヘッドを課するが、めったにシステム・バスを使用しないか、あるいは非常に低い帯域幅で用いるマスタに対しては有用である。

ANYRQSTがハイでなければ、CPUのローカル・バスがアイドルのとき( $\overline{\text{S2}}$ ,  $\overline{\text{S1}}$ ,  $\overline{\text{S0}} = 1, 1, 1$ )に限り、 $\overline{\text{CBRQ}}$ 要求のより低いプライオリティの素子に対して8289はバスを解放する。 $\overline{\text{CBRQ}}$ を無効にしてより低いプライオリティのマスタに対するバス解放を行なわないために、 $\overline{\text{CRQLCK}}$  (コモン・バス・リクエスト・ロック)がある。この入力をローにすることによって、入力としての $\overline{\text{CBRQ}}$ を有効にディスエーブルにできる。同様の効果は $\overline{\text{CBRQ}}$ をハイに接続することによって達成できるが、 $\overline{\text{CRQLCK}}$ によってスタティックな機能をプログラム可能な選択とすることができる。

$\overline{\text{CBRQ}}$ は双方向で、8289はバスを獲得するためにローに駆動し、バスを有するときはモニタすることに注意。したがって、バスを獲得するためにバスにまだ $\overline{\text{CBRQ}}$ を利用させている間に、8289が $\overline{\text{CBRQ}}$ を無視することを $\overline{\text{CRQLCK}}$ は認めている。

8289にバスを解放させる他の特殊な条件には、8289がI/Oバス(I/OB)モードに切り替えられている状態でのI/Oサイクル、8289がRESBモードとなっているレジデント・バス・サイクル、そして常にCPUがHALT命令の実行によってホルト状態に入るときがあげられる。ホルト状態に入るとは、ステータス・ラインのホルト・ステータス( $\overline{\text{S2}}$ ,  $\overline{\text{S1}}$ ,  $\overline{\text{S0}} = 0, 1, 1$ )によって8289に示される。

# 付録A

## 8086命令セット一覧

—アルファベット順—



| 命 令  | オブジェクト・コード                                                                     | バイト              | クロック                                                       |
|------|--------------------------------------------------------------------------------|------------------|------------------------------------------------------------|
| AAA  | 37                                                                             | 1                | 4                                                          |
| AAD  | D5<br>0A                                                                       | 2                | 60                                                         |
| AAM  | D4<br>0A                                                                       | 2                | 83                                                         |
| AAS  | 3F                                                                             | 1                | 4                                                          |
| ADC  | ac,data<br>0001010w<br>kk<br>[jj]                                              | 2 または 3          | 4                                                          |
| ADC  | mem/reg1,data<br>100000sw<br>mod 010 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj]     | 3, 4, 5<br>または 6 | reg: 4<br><br>mem: 17 + EA                                 |
| ADC  | mem/reg1,mem/reg2<br>000100dw<br>mod rrr r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj] | 2,3 または 4        | reg to reg: 3<br>mem to reg: 9 + EA<br>reg to mem: 16 + EA |
| ADD  | ac,data<br>0000010w<br>kk<br>[jj]                                              | 2 または 3          | 4                                                          |
| ADD  | mem/reg,data<br>100000sw<br>mod 000 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj]      | 3, 4, 5<br>または 6 | reg: 4<br>mem: 17 + EA                                     |
| ADD  | mem/reg1,mem/reg2<br>000000dw<br>mod rrr r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj] | 2,3 または 4        | reg to reg: 3<br>mem to reg: 9 + EA<br>reg to mem: 16 + EA |
| AND  | ac,data<br>0010010w<br>kk<br>[jj]                                              | 2 または 3          | 4                                                          |
| AND  | mem/reg,data<br>1000000w<br>mod 100 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj]      | 3, 4, 5 または<br>6 | reg: 4<br>mem: 17 + EA                                     |
| AND  | mem/reg1,mem/reg2<br>001000dw<br>mod rrr r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj] | 2,3 または 4        | reg to reg: 3<br>mem to reg: 9 + EA<br>reg to mem: 16 + EA |
| CALL | addr<br>9A<br>kk<br>jj<br>hh                                                   | 5                | 28                                                         |
| CALL | disp16<br>99<br>E8<br>kk<br>jj                                                 | 3                | 19                                                         |
| CALL | mem<br>FF<br>mod 011 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj]                     | 2,3 または 4        | 32-bit mem ポインタ:<br>37 + EA                                |
| CALL | mem/reg<br>FF<br>mod 010 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj]                 | 2,3 または 4        | 16-bit reg ポインタ:<br>16<br>16-bit mem ポインタ:<br>21 + EA      |

| 命 令  |                    | オブジェクト・コード                                                | バイト              | クロック                                                                                                                     |
|------|--------------------|-----------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------|
| CBW  |                    | 98                                                        | 1                | 2                                                                                                                        |
| CLC  |                    | F8                                                        | 1                | 2                                                                                                                        |
| CLD  |                    | FC                                                        | 1                | 2                                                                                                                        |
| CLI  |                    | FA                                                        | 1                | 2                                                                                                                        |
| CMC  |                    | F5                                                        | 1                | 2                                                                                                                        |
| CMP  | ac, data           | 0011110w<br>kk<br>[jj]                                    | 2 または 3          | 4                                                                                                                        |
| CMP  | mem/reg, data      | 100000sw<br>mod 111 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj] | 3, 4, 5 または<br>6 | reg: 4<br>mem: 10 + EA                                                                                                   |
| CMP  | mem/reg1, mem/reg2 | 001110dw<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | reg to reg: 3<br>mem to reg: 9 + EA<br>reg to mem: 9 + EA                                                                |
| CMPS |                    | 1010011w                                                  | 1                | 22<br>9 + 22/繰返し*                                                                                                        |
| CWD  |                    | 99                                                        | 1                | 5                                                                                                                        |
| DAA  |                    | 27                                                        | 1                | 4                                                                                                                        |
| DAS  |                    | 2F                                                        | 1                | 4                                                                                                                        |
| DEC  | mem/reg            | 1111111w<br>mod 001 r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | reg: 3<br>mem: 15 + EA                                                                                                   |
| DEC  | 16-bit reg         | 01001rrr                                                  | 1                | 2                                                                                                                        |
| DIV  | mem/reg            | 1111011w<br>mod 110 r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | 8-bit reg:<br>80 → 90<br>16-bit reg:<br>144 → 162<br>8-bit mem:<br>(86 → 96) + EA<br>16-bit mem:<br>(150 → 168) + EA     |
| ESC  | mem/reg            | 11011xxx<br>mod xxx r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | mem: 8 + EA<br>reg: 2                                                                                                    |
| HLT  |                    | F4                                                        | 1                | 2                                                                                                                        |
| IDIV | mem/reg            | 1111011w<br>mod 111 r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | 8-bit reg:<br>101 → 112<br>16-bit reg:<br>165 → 184<br>8-bit mem:<br>(107 → 118) + EA<br>16-bit mem:<br>(171 → 190) + EA |
| IMUL | mem/reg            | 1111011w<br>mod 101 r/m<br>[DISP]<br>[DISP]               | 2, 3 または 4       | 8-bit reg:<br>80 → 98<br>16-bit reg:<br>128 → 154<br>8-bit mem:<br>(86 → 104) + EA<br>16-bit mem:<br>(134 → 160) + EA    |
| IN   | ac, DX             | 1110110w                                                  | 1                | 8                                                                                                                        |
| IN   | ac, port           | 1110010w                                                  | 2                | 10                                                                                                                       |

\* REPプレフィックスが先行したとき

| 命 令  |            | オブジェクト・コード                                  | バイト       | クロック                                        |
|------|------------|---------------------------------------------|-----------|---------------------------------------------|
| INC  | mem/reg    | 1111111w<br>mod 000 r/m<br>[DISP]<br>[DISP] | 2,3 または 4 | reg: 3<br>mem: 15 + EA                      |
| INC  | 16-bit reg | 01000rrr                                    | 1         | 2                                           |
| INT  |            | 11001100*                                   | 1         | 52                                          |
|      |            | 11001101<br>type                            | 2         | 51                                          |
| INTO |            | CE                                          | 1         | interrupt: 53<br>no interrupt: 4            |
| IRET |            | CF                                          | 1         | 32                                          |
| JA   | disp       | 77                                          | 2         | 4/No Branch                                 |
| JNBE |            | disp                                        |           | 16/Branch                                   |
| JAE  | disp       | 73                                          | 2         | 4/No Branch                                 |
| JNB  |            | disp                                        |           | 16/Branch                                   |
| JB   | disp       | 72                                          | 2         | 4/No Branch                                 |
| JNAE |            | disp                                        |           | 8/Branch                                    |
| JBE  | disp       | 76                                          | 2         | 4/No Branch                                 |
| JNA  |            | disp                                        |           | 16/Branch                                   |
| JCXZ | disp       | E3                                          | 2         | 6/No Branch                                 |
|      |            | disp                                        |           | 18/Branch                                   |
| JE   | disp       | 74                                          | 2         | 4/No Branch                                 |
| JZ   |            | disp                                        |           | 16/Branch                                   |
| JG   | disp       | 7F                                          | 2         | 4/No Branch                                 |
| JNLE |            | disp                                        |           | 16/Branch                                   |
| JGE  | disp       | 7D                                          | 2         | 4/No Branch                                 |
| JNL  |            | disp                                        |           | 16/Branch                                   |
| JL   | disp       | 7C                                          | 2         | 4/No Branch                                 |
| JNGE |            | disp                                        |           | 16/Branch                                   |
| JLE  | disp       | 7E                                          | 2         | 4/No Branch                                 |
| JNG  |            | disp                                        |           | 16/Branch                                   |
| JMP  | addr       | EA<br>kk<br>jj<br>hh<br>99                  | 5         | 15                                          |
| JMP  | disp       | EB<br>disp                                  | 2         | 15                                          |
| JMP  | disp16     | E9<br>kk<br>jj                              | 3         | 15                                          |
| JMP  | mem        | FF<br>mod 101 r/m<br>[DISP]<br>[DISP]       | 2,3 または 4 | mem ptr 32:<br>24 + EA                      |
| JMP  | mem/reg    | FF<br>mod 100 rr/m<br>[DISP]<br>[DISP]      | 2,3 または 4 | reg ptr 16:<br>11<br>mem ptr 16:<br>16 + EA |
| JNE  | disp       | 75                                          | 2         | 4/No Branch                                 |
| JNZ  |            | disp                                        |           | 16/Branch                                   |
| JNO  | disp       | 71                                          | 2         | 4/No Branch                                 |
|      |            | disp                                        |           | 16/Branch                                   |
| JNP  | disp       | 7B                                          | 2         | 4/No Branch                                 |
| JPO  |            | disp                                        |           | 16/Branch                                   |
| JNS  | disp       | 79                                          | 2         | 4/No Branch                                 |
|      |            | disp                                        |           | 16/Branch                                   |
| JO   | disp       | 70                                          | 2         | 4/No Branch                                 |
|      |            | disp                                        |           | 16/Branch                                   |

\* type=3を仮定

| 命 令                   | オブジェクト・コード                                    | バイト           | クロック               |
|-----------------------|-----------------------------------------------|---------------|--------------------|
| JP disp               | 7A                                            | 2             | 4/No Branch        |
| JPE disp              | disp                                          |               | 16/Branch          |
| JS disp               | 78                                            | 2             | 4/No Branch        |
| LAHF                  | disp                                          |               | 16/Branch          |
| LDS reg,mem           | 9F                                            | 1             | 4                  |
|                       | C5                                            | 2,3 または 4     | 16 + EA            |
|                       | mod rrr r/m<br>[DISP]<br>[DISP]               |               |                    |
| LEA reg,mem           | 8D                                            | 2,3 または 4     | 2 + EA             |
|                       | mod rrr r/m<br>[DISP]<br>[DISP]               |               |                    |
| LES reg,mem           | C4                                            | 2,3 または 4     | 16 + EA            |
|                       | mod rrr r/m<br>[DISP]<br>[DISP]               |               |                    |
| LOCK                  | F0                                            | 1             | 2                  |
| LODS                  | 1010110w                                      | 1             | 12                 |
| LOOP disp             | E2                                            | 2             | 9 + 13/繰返し*        |
|                       | disp                                          |               | 5/No Branch        |
| LOOPE disp            | E1                                            | 2             | 17/Branch          |
| LOOPZ disp            | disp                                          |               | 6/No Branch        |
| LOOPNE disp           | E0                                            | 2             | 18/Branch          |
| LOOPNZ disp           | disp                                          |               | 5/No Branch        |
| MOV mem/reg1,mem/reg2 | 100010dw                                      | 2,3 または 4     | 19/Branch          |
|                       | mod rrr r/m<br>[DISP]<br>[DISP]               |               | reg to reg: 2      |
|                       |                                               |               | reg to mem: 8 + EA |
|                       |                                               |               | mem to reg: 9 + EA |
| MOV reg,data          | 1011wrrr<br>kk<br>[ij]                        | 2 または 3       | 4                  |
| MOV ac,mem            | 1010000w<br>kk<br>jj                          | 3             | 10                 |
| MOV mem,ac            | 1010001w<br>kk<br>jj                          | 3             | 10                 |
| MOV segreg,mem/reg    | 8E                                            | 2,3 または 4     | reg to reg: 2      |
|                       | mod 0rr r/m<br>[DISP]<br>[DISP]               |               | mem to reg: 8 + EA |
| MOV mem/reg,segreg    | 8C                                            | 2,3 または 4     | reg to reg: 2      |
|                       | mod 0rr r/m<br>[DISP]<br>[DISP]               |               | reg to mem: 9 + EA |
| MOV mem/reg,data      | 1100011w                                      | 3, 4, 5 または 6 | reg/mem: 10 + EA   |
|                       | mod 000 r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] |               |                    |
| MOVS                  | 1010010w                                      | 1             | 18                 |
|                       |                                               |               | 9 + 17/繰返し*        |

\* REPプレフィックスが先行したとき

| 命 令   |                   | オブジェクト・コード                                                | バイト              | クロック                                                                                                                 |
|-------|-------------------|-----------------------------------------------------------|------------------|----------------------------------------------------------------------------------------------------------------------|
| MUL   | mem/reg           | 1111011w<br>mod 100 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4        | 8-bit reg:<br>70 — 77<br>16-bit reg:<br>118 — 133<br>8-bit mem:<br>(76 — 83) + EA<br>16-bit mem:<br>(124 — 139) + EA |
| NEG   | mem/reg           | 1111011w<br>mod 011 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4        | reg: 3<br>mem: 16 + EA                                                                                               |
| NOP   |                   | 90                                                        | 1                | 3                                                                                                                    |
| NOT   | mem/reg           | 1111011w<br>mod 010 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4        | reg: 3<br>mem: 16 + EA                                                                                               |
| OR    | ac,data           | 0000110w<br>kk<br>[ij]                                    | 2 または 3          | 4                                                                                                                    |
| OR    | mem/reg,data      | 1000000w<br>mod 001 r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] | 3, 4, 5 または<br>6 | reg: 4<br>mem: 17 + EA                                                                                               |
| OR    | mem/reg1,mem/reg2 | 000010dw<br>mod rrr r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] | 3,4,5 または 6      | reg to reg: 3<br>mem to reg: 9 + EA<br>reg to mem: 16 + EA                                                           |
| OUT   | DX,ac             | 1110111w                                                  | 1                | 8                                                                                                                    |
| OUT   | port,ac           | 1110011w<br>yy                                            | 2                | 10                                                                                                                   |
| POP   | mem/reg           | 8F<br>mod 000 r/m<br>[DISP]<br>[DISP]                     | 2,3 または 4        | reg: 8<br>mem: 17 + EA                                                                                               |
| POP   | reg               | 01011rrr                                                  | 1                | 8                                                                                                                    |
| POP   | segreg            | 000ss111                                                  | 1                | 8                                                                                                                    |
| POPF  |                   | 9D                                                        | 1                | 8                                                                                                                    |
| PUSH  | mem/reg           | FF<br>mod 110 r/m<br>[DISP]<br>[DISP]                     | 2,3 または 4        | reg: 11<br>mem: 16 + EA                                                                                              |
| PUSH  | reg               | 01010rrr                                                  | 1                | 10                                                                                                                   |
| PUSH  | segreg            | 000ss110                                                  | 1                | 10                                                                                                                   |
| PUSHF |                   | 9C                                                        | 1                | 10                                                                                                                   |
| RCL   | mem/reg,count     | 110100cw<br>mod 010 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4        | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N)                    |

NはCL内の値



| 命 令                                               | オブジェクト・コード                                                | バイト           | クロック                                                                                              |
|---------------------------------------------------|-----------------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------|
| RCR<br>mem/reg,count                              | 110100cw<br>mod 011 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |
| REP<br>/REPE/REPNE                                | 1111001z                                                  | 1             | 2                                                                                                 |
| RET<br>(セグメント間)                                   | CB                                                        | 1             | 24                                                                                                |
| RET<br>(セグメント内)                                   | C3                                                        | 1             | 16                                                                                                |
| RET<br>disp16 (セグメント間)                            | CA<br>kk<br>jj                                            | 3             | 23                                                                                                |
| RET<br>disp16 (セグメント内)                            | C2<br>kk<br>jj                                            | 3             | 20                                                                                                |
| ROL<br>mem/reg,count                              | 110100cw<br>mod 000 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |
| ROR<br>mem/reg,count                              | 110100cw<br>mod 001 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |
| SAHF                                              | 9E                                                        | 1             | 4                                                                                                 |
| SAR<br>mem/reg,count                              | 110100cw<br>mod 111 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |
| SBB<br>ac,data                                    | 0001110w<br>kk<br>[jj]                                    | 2 または 3       | 4                                                                                                 |
| SBB<br>mem/reg,data                               | 100000sw<br>mod 011 r/m<br>[DISP]<br>[DISP]<br>kk<br>[jj] | 3, 4, 5 または 6 | reg: 4<br>mem: 17 + EA                                                                            |
| SBB<br>mem/reg <sub>1</sub> ,mem/reg <sub>2</sub> | 000110dw<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | reg from reg: 3<br>mem from reg: 9 + EA<br>reg.from mem: 16 + EA                                  |
| SCAS                                              | 1010111w                                                  | 1             | 15<br>9 + 15/繰返し*                                                                                 |
| SEG                                               | 001ss110                                                  | 1             | 2                                                                                                 |
| SHL<br>mem/reg,count                              | 110100cw<br>mod 100 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |

\* REPプレフィックスが先行したとき  
NはCL内の値

| 命 令                                               | オブジェクト・コード                                                | バイト           | クロック                                                                                              |
|---------------------------------------------------|-----------------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------|
| SHR<br>mem/reg.count                              | 110100cw<br>mod 101 r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | count = 1<br>reg: 2<br>mem: 15 + EA<br>count = [CL]<br>reg: 8 + (4 * N)<br>mem: 20 + EA + (4 * N) |
| STC                                               | F9                                                        | 1             | 2                                                                                                 |
| STD                                               | FD                                                        | 1             | 2                                                                                                 |
| STI                                               | FB                                                        | 1             | 2                                                                                                 |
| STOS                                              | 1010101w                                                  | 1             | 11<br>9 + 10/繰返し*                                                                                 |
| SUB<br>ac,data                                    | 0010110w<br>kk<br>[ij]                                    | 2 または 3       | 4                                                                                                 |
| SUB<br>mem/reg.data                               | 100000sw<br>mod 101 r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] | 3, 4, 5 または 6 | reg: 4<br>mem: 17 + EA                                                                            |
| SUB<br>mem/reg <sub>1</sub> ,mem/reg <sub>2</sub> | 001010dw<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | reg from reg: 3<br>mem from reg: 9 + EA<br>reg from mem: 16 + EA                                  |
| TEST<br>ac,data                                   | 1010100w<br>kk<br>[ij]                                    | 2 または 3       | 4                                                                                                 |
| TEST<br>mem/reg.data                              | 1111011w<br>mod 000 r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] | 3, 4, 5 または 6 | reg: 5<br>mem: 11 + EA                                                                            |
| TEST<br>reg.mem/reg                               | 1000010w<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | reg with reg: 3<br>reg with mem: 9 + EA                                                           |
| WAIT                                              | 9B                                                        | 1             | 3(最小) + 5n                                                                                        |
| XCHG<br>reg,ac                                    | 10010rrr                                                  | 1             | 3                                                                                                 |
| XCHG<br>reg.mem/reg                               | 1000011w<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | reg with reg: 4<br>reg with mem: 17 + EA                                                          |
| XLAT                                              | D7                                                        | 1             | 11                                                                                                |
| XOR<br>ac,data                                    | 0011010w<br>kk<br>[ij]                                    | 2 または 3       | 4                                                                                                 |
| XOR<br>mem/reg.data                               | 1000000w<br>mod 110 r/m<br>[DISP]<br>[DISP]<br>kk<br>[ij] | 3, 4, 5 または 6 | reg: 4<br>mem: 17 + EA                                                                            |
| XOR<br>mem/reg <sub>1</sub> ,mem/reg <sub>2</sub> | 001100dw<br>mod rrr r/m<br>[DISP]<br>[DISP]               | 2,3 または 4     | reg with reg: 3<br>mem with reg: 9 + EA<br>reg with mem: 16 + EA                                  |

\* REPプレフィックスが先行したとき

nは、TESTインプットのサンプル当たりのクロック

# 付録B

## 8086命令セット一覧

—オブジェクト・コード数値上昇順—

| オブジェクト・コード |             |              | ニーモニック                 |
|------------|-------------|--------------|------------------------|
| バイト# 0     | バイト# 1      | 後続バイト        |                        |
| 00         | mod reg r/m | [disp][disp] | ADD mem/reg,reg (バイト)  |
| 01         | mod reg r/m | [disp][disp] | ADD mem/reg,reg (ワード)  |
| 02         | mod reg r/m | [disp][disp] | ADD reg, mem/reg (バイト) |
| 03         | mod reg r/m | [disp][disp] | ADD reg, mem/reg (ワード) |
| 04         | kk          |              | ADD AL, kk             |
| 05         | kk          | ij           | ADD AX, ijkk           |
| 06         |             |              | PUSH ES                |
| 07         |             |              | POP ES                 |
| 08         | mod reg r/m | [disp][disp] | OR mem/reg,reg (バイト)   |
| 09         | mod reg r/m | [disp][disp] | OR mem/reg,reg (ワード)   |
| 0A         | mod reg r/m | [disp][disp] | OR reg, mem/reg (バイト)  |
| 0B         | mod reg r/m | [disp][disp] | OR reg, mem/reg (ワード)  |
| 0C         | kk          |              | OR AL, kk              |
| 0D         | kk          | ij           | OR AX, ijkk            |
| 0E         |             |              | PUSH CS                |
| 0F         |             |              | 未使用                    |
| 10         | mod reg r/m | [disp][disp] | ADC mem/reg,reg (バイト)  |
| 11         | mod reg r/m | [disp][disp] | ADC mem/reg,reg (ワード)  |
| 12         | mod reg r/m | [disp][disp] | ADC reg, mem/reg (バイト) |
| 13         | mod reg r/m | [disp][disp] | ADC reg, mem/reg (ワード) |
| 14         | kk          |              | ADC AL, kk             |
| 15         | kk          | ij           | ADC AX, ijkk           |
| 16         |             |              | PUSH SS                |
| 17         |             |              | POP SS                 |
| 18         | mod reg r/m | [disp][disp] | SBB mem/reg,reg (バイト)  |
| 19         | mod reg r/m | [disp][disp] | SBB mem/reg,reg (ワード)  |
| 1A         | mod reg r/m | [disp][disp] | SBB reg, mem/reg (バイト) |
| 1B         | mod reg r/m | [disp][disp] | SBB reg, mem/reg (ワード) |
| 1C         | kk          |              | SBB AL, kk             |
| 1D         | kk          | ij           | SBB AX, ijkk           |
| 1E         |             |              | PUSH DS                |
| 1F         |             |              | POP DS                 |
| 20         | mod reg r/m | [disp][disp] | AND mem/reg,reg (バイト)  |
| 21         | mod reg r/m | [disp][disp] | AND mem/reg,reg (ワード)  |
| 22         | mod reg r/m | [disp][disp] | AND reg, mem/reg (バイト) |
| 23         | mod reg r/m | [disp][disp] | AND reg, mem/reg (ワード) |
| 24         | kk          |              | AND AL, kk             |
| 25         | kk          | ij           | AND AX, ijkk           |
| 26         |             |              | SEG ES                 |
| 27         |             |              | DAA                    |
| 28         | mod reg r/m | [disp][disp] | SUB mem/reg,reg (バイト)  |
| 29         | mod reg r/m | [disp][disp] | SUB mem/reg,reg (ワード)  |
| 2A         | mod reg r/m | [disp][disp] | SUB reg, mem/reg (バイト) |
| 2B         | mod reg r/m | [disp][disp] | SUB reg, mem/reg (ワード) |
| 2C         | kk          |              | SUB AL, kk             |
| 2D         | kk          | ij           | SUB AX, ijkk           |
| 2E         |             |              | SEG CS                 |
| 2F         |             |              | DAS                    |

| オブジェクト・コード |             |              | ニーモニック                |
|------------|-------------|--------------|-----------------------|
| バイト # 0    | バイト # 1     | 後続バイト        |                       |
| 30         | mod reg r/m | [disp][disp] | XOR mem/reg,reg (バイト) |
| 31         | mod reg r/m | [disp][disp] | XOR mem/reg,reg (ワード) |
| 32         | mod reg r/m | [disp][disp] | XOR reg,mem/reg (バイト) |
| 33         | mod reg r/m | [disp][disp] | XOR reg,mem/reg (ワード) |
| 34         | kk          |              | XOR AL,kk             |
| 35         | kk          | jj           | XOR AX,jkk            |
| 36         |             |              | SEG SS                |
| 37         |             |              | AAA                   |
| 38         | mod reg r/m | [disp][disp] | CMP mem/reg,reg (バイト) |
| 39         | mod reg r/m | [disp][disp] | CMP mem/reg,reg (ワード) |
| 3A         | mod reg r/m | [disp][disp] | CMP reg,mem/reg (バイト) |
| 3B         | mod reg r/m | [disp][disp] | CMP reg,mem/reg (ワード) |
| 3C         | kk          |              | CMP AL,kk             |
| 3D         | kk          | jj           | CMP AX,jkk            |
| 3E         |             |              | SEG DS                |
| 3F         |             |              | AAS                   |
| 40         |             |              | INC AX                |
| 41         |             |              | INC CX                |
| 42         |             |              | INC DX                |
| 43         |             |              | INC BX                |
| 44         |             |              | INC SP                |
| 45         |             |              | INC BP                |
| 46         |             |              | INC SI                |
| 47         |             |              | INC DI                |
| 48         |             |              | DEC AX                |
| 49         |             |              | DEC CX                |
| 4A         |             |              | DEC DX                |
| 4B         |             |              | DEC BX                |
| 4C         |             |              | DEC SP                |
| 4D         |             |              | DEC BP                |
| 4E         |             |              | DEC SI                |
| 4F         |             |              | DEC DI                |
| 50         |             |              | PUSH AX               |
| 51         |             |              | PUSH CX               |
| 52         |             |              | PUSH DX               |
| 53         |             |              | PUSH BX               |
| 54         |             |              | PUSH SP               |
| 55         |             |              | PUSH BP               |
| 56         |             |              | PUSH SI               |
| 57         |             |              | PUSH DI               |
| 58         |             |              | POP AX                |
| 59         |             |              | POP CX                |
| 5A         |             |              | POP DX                |
| 5B         |             |              | POP BX                |
| 5C         |             |              | POP SP                |
| 5D         |             |              | POP BP                |
| 5E         |             |              | POP SI                |
| 5F         |             |              | POP DI                |
| 60-6F      |             |              | 未使用                   |



| オブジェクト・コード |             |                   | ニーモニック                      |
|------------|-------------|-------------------|-----------------------------|
| バイト # 0    | バイト # 1     | 後続バイト             |                             |
| 70         | disp        |                   | JO disp                     |
| 71         | disp        |                   | JNO disp                    |
| 72         | disp        |                   | JB or JNAE or JC disp       |
| 73         | disp        |                   | JNB or JAE or JNC disp      |
| 74         | disp        |                   | JE or JZ disp               |
| 75         | disp        |                   | JNE or JNZ disp             |
| 76         | disp        |                   | JBE or JNA disp             |
| 77         | disp        |                   | JNBE or JA disp             |
| 78         | disp        |                   | JS disp                     |
| 79         | disp        |                   | JNS disp                    |
| 7A         | disp        |                   | JP or JPE disp              |
| 7B         | disp        |                   | JNP or JPO disp             |
| 7C         | disp        |                   | JL or JNGE disp             |
| 7D         | disp        |                   | JNL or JGE disp             |
| 7E         | disp        |                   | JLE or JNG disp             |
| 7F         | disp        |                   | JNLE or JG disp             |
| 80         | mod 000 r/m | [disp][disp] kk   | ADD mem/reg, kk             |
| 80         | mod 001 r/m | [disp][disp] kk   | OR mem/reg, kk              |
| 80         | mod 010 r/m | [disp][disp] kk   | ADC mem/reg, kk             |
| 80         | mod 011 r/m | [disp][disp] kk   | SBB mem/reg, kk             |
| 80         | mod 100 r/m | [disp][disp] kk   | AND mem/reg, kk             |
| 80         | mod 101 r/m | [disp][disp] kk   | SUB mem/reg, kk             |
| 80         | mod 110 r/m | [disp][disp] kk   | XOR mem/reg, kk             |
| 80         | mod 111 r/m | [disp][disp] kk   | CMP mem/reg, kk             |
| 81         | mod 000 r/m | [disp][disp] kkjj | ADD mem/reg, jkk            |
| 81         | mod 001 r/m | [disp][disp] kkjj | OR mem/reg, jkk             |
| 81         | mod 010 r/m | [disp][disp] kkjj | ADC mem/reg, jkk            |
| 81         | mod 011 r/m | [disp][disp] kkjj | SBB mem/reg, jkk            |
| 81         | mod 100 r/m | [disp][disp] kkjj | AND mem/reg, jkk            |
| 81         | mod 101 r/m | [disp][disp] kkjj | SUB mem/reg, jkk            |
| 81         | mod 110 r/m | [disp][disp] kkjj | XOR mem/reg, jkk            |
| 81         | mod 111 r/m | [disp][disp] kkjj | CMP mem/reg, jkk            |
| 82         | mod 000 r/m | [disp][disp] kk   | ADD mem/reg, kk (バイト)       |
| 82         | xx 001 xxx  |                   | 未使用                         |
| 82         | mod 010 r/m | [disp][disp] kk   | ADC mem/reg, kk (バイト)       |
| 82         | mod 011 r/m | [disp][disp] kk   | SBB mem/reg, kk (バイト)       |
| 82         | xx 100 xxx  |                   | 未使用                         |
| 82         | mod 101 r/m | [disp][disp] kk   | SUB mem/reg, kk (バイト)       |
| 82         | xx 110 xxx  |                   | 未使用                         |
| 82         | mod 111 r/m | [disp][disp] kk   | CMP mem/reg, kk (バイト)       |
| 83         | mod 000 r/m | [disp][disp] kk   | ADD mem/reg, jkk (ワードに符号拡張) |
| 83         | xx 001 xxx  |                   | 未使用                         |
| 83         | mod 010 r/m | [disp][disp] kk   | ADC mem/reg, jkk (ワードに符号拡張) |
| 83         | mod 011 r/m | [disp][disp] kk   | SBB mem/reg, jkk (ワードに符号拡張) |
| 83         | xx 100 r/m  |                   | 未使用                         |
| 83         | mod 101 r/m | [disp][disp] kk   | SUB mem/reg, jkk (ワードに符号拡張) |
| 83         | xx 110 xxx  |                   | 未使用                         |
| 83         | mod 111 r/m | [disp][disp] kk   | CMP mem/reg, jkk (ワードに符号拡張) |
| 84         | mod reg r/m | [disp][disp]      | TEST mem/reg, reg (バイト)     |
| 85         | mod reg r/m | [disp][disp]      | TEST mem/reg, reg (ワード)     |
| 86         | mod reg r/m | [disp][disp]      | XCHG reg, mem/reg (バイト)     |
| 87         | mod reg r/m | [disp][disp]      | XCHG reg, mem/reg (ワード)     |
| 88         | mod reg r/m | [disp][disp]      | MOV mem/reg, reg (バイト)      |
| 89         | mod reg r/m | [disp][disp]      | MOV mem/reg, reg (ワード)      |

| オブジェクト・コード |             |              | ニーモニック                |
|------------|-------------|--------------|-----------------------|
| バイト # 0    | バイト # 1     | 後続バイト        |                       |
| 8A         | mod reg r/m | [disp][disp] | MOV reg,mem/reg (バイト) |
| 8B         | mod reg r/m | [disp][disp] | MOV reg,mem/reg (ワード) |
| 8C         | mod 0ss r/m | [disp][disp] | MOV mem/reg,segreg    |
| 8C         | xx 1xxxxx   |              | 未使用                   |
| 8D         | mod reg r/m | [disp][disp] | LEA reg,addr          |
| 8E         | mod 0ss r/m | [disp][disp] | MOV segreg, mem/reg   |
| 8E         | xx 1xxxxx   |              | 未使用                   |
| 8F         | mod 000 r/m | [disp][disp] | POP mem/reg           |
| 8F         | xx 001 xxx  |              | 未使用                   |
| 8F         | xx 010 xxx  |              | 未使用                   |
| 8F         | xx 011 xxx  |              | 未使用                   |
| 8F         | xx 100 xxx  |              | 未使用                   |
| 8F         | xx 101 xxx  |              | 未使用                   |
| 8F         | xx 110 xxx  |              | 未使用                   |
| 8F         | xx 111 xxx  |              | 未使用                   |
| 90         |             |              | NOP                   |
| 91         |             |              | XCHG AX,CX            |
| 92         |             |              | XCHG AX,DX            |
| 93         |             |              | XCHG AX,BX            |
| 94         |             |              | XCHG AX,SP            |
| 95         |             |              | XCHG AX,BP            |
| 96         |             |              | XCHG AX,SI            |
| 97         |             |              | XCHG AX,DI            |
| 98         |             |              | CBW                   |
| 99         |             |              | CWD                   |
| 9A         | kk          | jj hh gg     | CALL addr             |
| 9B         |             |              | WAIT                  |
| 9C         |             |              | PUSHF                 |
| 9D         |             |              | POPF                  |
| 9E         |             |              | SAHF                  |
| 9F         |             |              | LAHF                  |
| A0         | qq          | pp           | MOV AL,addr           |
| A1         | qq          | pp           | MOV AX,addr           |
| A2         | qq          | pp           | MOV addr,AL           |
| A3         | qq          | pp           | MOV addr,AX           |
| A4         |             |              | MOVS BYTE             |
| A5         |             |              | MOVS WORD             |
| A6         |             |              | CMPS BYTE             |
| A7         |             |              | CMPS WORD             |
| A8         | kk          |              | TEST, AL,kk           |
| A9         | kk          | jj           | TEST AX,ijkk          |
| AA         |             |              | STOS BYTE             |
| AB         |             |              | STOS WORD             |
| AC         |             |              | LODS BYTE             |
| AD         |             |              | LODS WORD             |
| AE         |             |              | SCAS BYTE             |
| AF         |             |              | SCAS WORD             |

| オブジェクト・コード |             |                   | ニーモニック        |
|------------|-------------|-------------------|---------------|
| バイト# 0     | バイト# 1      | 後続バイト             |               |
| B0         | kk          |                   | MOV AL, kk    |
| B1         | kk          |                   | MOV CL, kk    |
| B2         | kk          |                   | MOV DL, kk    |
| B3         | kk          |                   | MOV BL, kk    |
| B4         | kk          |                   | MOV AH, kk    |
| B5         | kk          |                   | MOV CH, kk    |
| B6         | kk          |                   | MOV DH, kk    |
| B7         | kk          |                   | MOV BH, kk    |
| B8         | kk          | ij                | MOV AX, ijkk  |
| B9         | kk          | ij                | MOV CX, ijkk  |
| BA         | kk          | ij                | MOV DX, ijkk  |
| BB         | kk          | ij                | MOV BX, ijkk  |
| BC         | kk          | ij                | MOV SP, ijkk  |
| BD         | kk          | ij                | MOV BP, ijkk  |
| BE         | kk          | ij                | MOV SI, ijkk  |
| BF         | kk          | ij                | MOV DI, ijkk  |
| C0         |             |                   | 未使用           |
| C1         |             |                   | 未使用           |
| C2         | kk          | ij                | RET ijkk      |
| C3         |             |                   | RET           |
| C4         | mod reg r/m | [disp][disp]      | LES reg, addr |
| C5         | mod reg r/m | [disp][disp]      | LDS reg, addr |
| C6         | mod 000 r/m | [disp][disp] kk   | MOV mem, kk   |
| C6         | xx 001 xxx  |                   | 未使用           |
| C6         | xx 010 xxx  |                   | 未使用           |
| C6         | xx 011 xxx  |                   | 未使用           |
| C6         | xx 100 xxx  |                   | 未使用           |
| C6         | xx 101 xxx  |                   | 未使用           |
| C6         | xx 110 xxx  |                   | 未使用           |
| C6         | xx 111 xxx  |                   | 未使用           |
| C7         | mod 000 r/m | [disp][disp] kkij | MOV mem, ijkk |
| C7         | xx 001 xxx  |                   | 未使用           |
| C7         | xx 010 xxx  |                   | 未使用           |
| C7         | xx 011 xxx  |                   | 未使用           |
| C7         | xx 100 xxx  |                   | 未使用           |
| C7         | xx 101 xxx  |                   | 未使用           |
| C7         | xx 110 xxx  |                   | 未使用           |
| C7         | xx 111 xxx  |                   | 未使用           |
| C8         |             |                   | 未使用           |
| C9         |             |                   | 未使用           |
| CA         | kk          | ij                | RET ijkk      |
| CB         |             |                   | RET           |
| CC         |             |                   | INT 3         |
| CD         | type        |                   | INT Type      |
| CE         |             |                   | INTO          |
| CF         |             |                   | IRET          |

| オブジェクト・コード |             |              | ニーモニック                     |
|------------|-------------|--------------|----------------------------|
| バイト # 0    | バイト # 1     | 後続バイト        |                            |
| D0         | mod 000 r/m | [disp][disp] | ROL mem/reg,1(バイト)         |
| D0         | mod 001 r/m | [disp][disp] | ROR mem/reg,1(バイト)         |
| D0         | mod 010 r/m | [disp][disp] | RCL mem/reg,1(バイト)         |
| D0         | mod 011 r/m | [disp][disp] | RCR mem/reg,1(バイト)         |
| D0         | mod 100 r/m | [disp][disp] | SAL or SHL mem/reg,1(バイト)  |
| D0         | mod 101 r/m | [disp][disp] | SHR mem/reg,1(バイト)         |
| D0         | xx 110 xxx  |              | 未使用                        |
| D0         | mod 111 r/m | [disp][disp] | SAR mem/reg,1(バイト)         |
| D1         | mod 000 r/m | [disp][disp] | ROL mem/reg,1(ワード)         |
| D1         | mod 001 r/m | [disp][disp] | ROR mem/reg,1(ワード)         |
| D1         | mod 010 r/m | [disp][disp] | RCL mem/reg,1(ワード)         |
| D1         | mod 011 r/m | [disp][disp] | RCR mem/reg,1(ワード)         |
| D1         | mod 100 r/m | [disp][disp] | SAL or SHL mem/reg,1(ワード)  |
| D1         | mod 101 r/m | [disp][disp] | SHR mem/reg,1(ワード)         |
| D1         | xx 110 xxx  |              | 未使用                        |
| D1         | mod 111 r/m | [disp][disp] | SAR mem/reg,1(ワード)         |
| D2         | mod 000 r/m | [disp][disp] | ROL mem/reg,CL(バイト)        |
| D2         | mod 001 r/m | [disp][disp] | ROR mem/reg,CL(バイト)        |
| D2         | mod 010 r/m | [disp][disp] | RCL mem/reg,CL(バイト)        |
| D2         | mod 011 r/m | [disp][disp] | RCR mem/reg,CL(バイト)        |
| D2         | mod 100 r/m | [disp][disp] | SAL or SHL mem/reg,CL(バイト) |
| D2         | mod 101 r/m | [disp][disp] | SHR mem/reg,CL(バイト)        |
| D2         | xx 110 xxx  |              | 未使用                        |
| D2         | mod 111 r/m | [disp][disp] | SAR mem/reg,CL(バイト)        |
| D3         | mod 000 r/m | [disp][disp] | ROL mem/reg,CL(ワード)        |
| D3         | mod 001 r/m | [disp][disp] | ROR mem/reg,CL(ワード)        |
| D3         | mod 010 r/m | [disp][disp] | RCL mem/reg,CL(ワード)        |
| D3         | mod 011 r/m | [disp][disp] | RCR mem/reg,CL(ワード)        |
| D3         | mod 100 r/m | [disp][disp] | SAL or SHL mem/reg,CL(ワード) |
| D3         | mod 101 r/m | [disp][disp] | SHR mem/reg,CL(ワード)        |
| D3         | xx 110 xxx  |              | 未使用                        |
| D3         | mod 111 r/m | [disp][disp] | SAR mem/reg,CL(ワード)        |
| D4         | 0A          |              | AAM                        |
| D5         | 0A          |              | AAD                        |
| D6         |             |              | 未使用                        |
| D7         |             |              | XLAT                       |
| D8         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| D9         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DA         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DB         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DC         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DD         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DE         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| DF         | mod xxx r/m | [disp][disp] | ESC mem/reg                |
| E0         | disp        |              | LOOPNE/LOOPNZ disp         |
| E1         | disp        |              | LOOPE/LOOPZ disp           |
| E2         | disp        |              | LOOP disp                  |
| E3         | disp        |              | JCXZ disp                  |
| E4         | kk          |              | IN AL,kk                   |
| E5         | kk          |              | IN AX,kk                   |
| E6         | kk          |              | OUT kk,AL                  |
| E7         | kk          |              | OUT kk,AX                  |
| E8         | disp        | disp         | CALL disp16                |
| E9         | disp        | disp         | JMP disp16                 |

| オブジェクト・コード |             |                   | ニーモニック              |
|------------|-------------|-------------------|---------------------|
| バイト # 0    | バイト # 1     | 後続バイト             |                     |
| EA         | kk          | jj hh gg          | JMP addr            |
| EB         | disp        |                   | JMP disp            |
| EC         |             |                   | IN AL,DX            |
| ED         |             |                   | IN AX,DX            |
| EE         |             |                   | OUT DX,AL           |
| EF         |             |                   | OUT DX,AX           |
| F0         |             |                   | LOCK                |
| F1         |             |                   | 未使用                 |
| F2         |             |                   | REPNE or REPNZ      |
| F3         |             |                   | REP or REPE or REPZ |
| F4         |             |                   | HLT                 |
| F5         |             |                   | CMC                 |
| F6         | mod 000 r/m | [disp][disp] kk   | TEST mem/reg, kk    |
| F6         | xx 001 xxx  |                   | 未使用                 |
| F6         | mod 010 r/m | [disp][disp]      | NOT mem/reg (バイト)   |
| F6         | mod 011 r/m | [disp][disp]      | NEG mem/reg (バイト)   |
| F6         | mod 100 r/m | [disp][disp]      | MUL mem/reg (バイト)   |
| F6         | mod 101 r/m | [disp][disp]      | IMUL mem/reg (バイト)  |
| F6         | mod 110 r/m | [disp][disp]      | DIV mem/reg (バイト)   |
| F6         | mod 111 r/m | [disp][disp]      | IDIV mem/reg (バイト)  |
| F7         | mod 000 r/m | [disp][disp] kkjj | TEST mem/reg, jikk  |
| F7         | xx 001 xxx  |                   | 未使用                 |
| F7         | mod 010 r/m | [disp][disp]      | NOT mem/reg (ワード)   |
| F7         | mod 011 r/m | [disp][disp]      | NEG mem/reg (ワード)   |
| F7         | mod 100 r/m | [disp][disp]      | MUL mem/reg (ワード)   |
| F7         | mod 101 r/m | [disp][disp]      | IMUL mem/reg (ワード)  |
| F7         | mod 110 r/m | [disp][disp]      | DIV mem/reg (ワード)   |
| F7         | mod 111 r/m | [disp][disp]      | IDIV mem/reg (ワード)  |
| F8         |             |                   | CLC                 |
| F9         |             |                   | STC                 |
| FA         |             |                   | CLI                 |
| FB         |             |                   | STI                 |
| FC         |             |                   | CLD                 |
| FD         |             |                   | STD                 |
| FE         | mod 000 r/m | [disp][disp]      | INC mem/reg (バイト)   |
| FE         | mod 001 r/m | [disp][disp]      | DEC mem/reg (バイト)   |
| FE         | xx 010 xxx  |                   | 未使用                 |
| FE         | xx 011 xxx  |                   | 未使用                 |
| FE         | xx 100 xxx  |                   | 未使用                 |
| FE         | xx 101 xxx  |                   | 未使用                 |
| FE         | xx 110 xxx  |                   | 未使用                 |
| FE         | xx 111 xxx  |                   | 未使用                 |
| FF         | mod 000 r/m | [disp][disp]      | INC mem/reg (ワード)   |
| FF         | mod 001 r/m | [disp][disp]      | DEC mem/reg (ワード)   |
| FF         | mod 010 r/m | [disp][disp]      | CALL mem/reg        |
| FF         | mod 011 r/m | [disp][disp]      | CALL mem            |
| FF         | mod 100 r/m | [disp][disp]      | JMP mem/reg         |
| FF         | mod 101 r/m | [disp][disp]      | JMP mem             |
| FF         | mod 110 r/m | [disp][disp]      | PUSH mem            |
| FF         | xx 111 xxx  |                   | 未使用                 |



# 付録C

## 8086と8088ファミリーの AC, DC特性と信号波形

ここでは、原著のものではなく、Intelの“Component Data Catalog”, January 1982から許可を得て、訳者が判断して選択したものが掲載されている。

掲載内容は以下に示す素子についての電気的特性とタイミング・データであり、ほぼ原著のものと同じである。

- 8086CPU
- 8088CPU
- 8282/8283オクタル・ラッチ
- 8284Aクロック・ジェネレータ
- 8286/8287オクタル・バス・トランシーバ
- 8288バス・コントローラ
- 8289バス・アービタ
- その他

## iAPX 86/10 16-BIT HMOS MICROPROCESSOR

8086/8086-2/8086-1

- Direct Addressing Capability to 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages.
- 14 Word, by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Bit, Byte, Word, and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide
- Range of Clock Rates:  
5 MHz for 8086,  
8 MHz for 8086-2,  
10 MHz for 8086-1
- MULTIBUS™ System Compatible Interface

The Intel iAPX 86/10 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The iAPX 86/10 operates in both single processor and multiple processor configurations to achieve high performance levels.

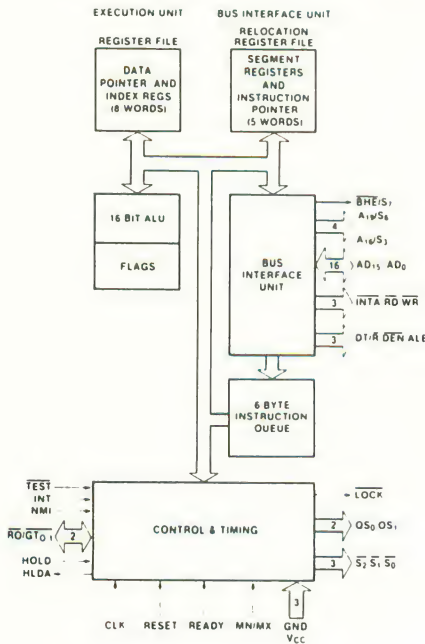
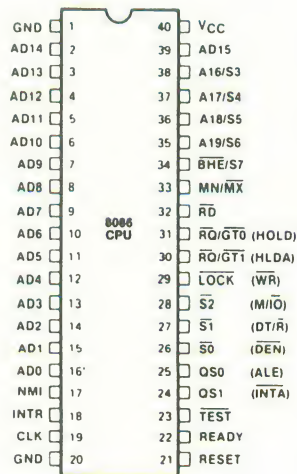


Figure 1. iAPX 86/10 CPU Block Diagram



40 LEAD

Figure 2. iAPX 86/10 Pin Configuration

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... - 65°C to + 150°C  
 Voltage on Any Pin with  
 Respect to Ground ..... - 1.0 to + 7V  
 Power Dissipation ..... 2.5 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**D.C. CHARACTERISTICS** (8086:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 10\%$ )  
 (8086-1:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )  
 (8086-2:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )

| Symbol   | Parameter                                                                                              | Min.  | Max.              | Units         | Test Conditions                         |
|----------|--------------------------------------------------------------------------------------------------------|-------|-------------------|---------------|-----------------------------------------|
| $V_{IL}$ | Input Low Voltage                                                                                      | - 0.5 | + 0.8             | V             |                                         |
| $V_{IH}$ | Input High Voltage                                                                                     | 2.0   | $V_{CC} + 0.5$    | V             |                                         |
| $V_{OL}$ | Output Low Voltage                                                                                     |       | 0.45              | V             | $I_{OL} = 2.5\text{ mA}$                |
| $V_{OH}$ | Output High Voltage                                                                                    | 2.4   |                   | V             | $I_{OH} = - 400\text{ }\mu\text{A}$     |
| $I_{CC}$ | Power Supply Current: 8086<br>8086-1<br>8086-2                                                         |       | 340<br>360<br>350 | mA            | $T_A = 25^\circ\text{C}$                |
| $I_{LI}$ | Input Leakage Current                                                                                  |       | $\pm 10$          | $\mu\text{A}$ | $0\text{V} \leq V_{IN} \leq V_{CC}$     |
| $I_{LO}$ | Output Leakage Current                                                                                 |       | $\pm 10$          | $\mu\text{A}$ | $0.45\text{V} \leq V_{OUT} \leq V_{CC}$ |
| $V_{CL}$ | Clock Input Low Voltage                                                                                | - 0.5 | + 0.6             | V             |                                         |
| $V_{CH}$ | Clock Input High Voltage                                                                               | 3.9   | $V_{CC} + 1.0$    | V             |                                         |
| $C_{IN}$ | Capacitance of Input Buffer<br>(All input except<br>$AD_0 - AD_{15}$ , $\overline{RQ}/\overline{GT}$ ) |       | 15                | pF            | $f_c = 1\text{ MHz}$                    |
| $C_{IO}$ | Capacitance of I/O Buffer<br>( $AD_0 - AD_{15}$ , $\overline{RQ}/\overline{GT}$ )                      |       | 15                | pF            | $f_c = 1\text{ MHz}$                    |

**A.C. CHARACTERISTICS** (8086:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 10\%$ )  
 (8086-1:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )  
 (8086-2:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )

**MINIMUM COMPLEXITY SYSTEM  
TIMING REQUIREMENTS**

| Symbol  | Parameter                                  | 8086 |      | 8086-1 (Preliminary) |      | 8086-2 |      | Units | Test Conditions   |
|---------|--------------------------------------------|------|------|----------------------|------|--------|------|-------|-------------------|
|         |                                            | Min. | Max. | Min.                 | Max. | Min.   | Max. |       |                   |
| TCLCL   | CLK Cycle Period                           | 200  | 500  | 100                  | 500  | 125    | 500  | ns    |                   |
| TCLCH   | CLK Low Time                               | 118  |      | 53                   |      | 68     |      | ns    |                   |
| TCHCL   | CLK High Time                              | 69   |      | 39                   |      | 44     |      | ns    |                   |
| TCH1CH2 | CLK Rise Time                              |      | 10   |                      | 10   |        | 10   | ns    | From 1.0V to 3.5V |
| TCL2CL1 | CLK Fall Time                              |      | 10   |                      | 10   |        | 10   | ns    | From 3.5V to 1.0V |
| TDVCL   | Data in Setup Time                         | 30   |      | 5                    |      | 20     |      | ns    |                   |
| TCLDX   | Data in Hold Time                          | 10   |      | 10                   |      | 10     |      | ns    |                   |
| TR1VCL  | RDY Setup Time into 8284A (See Notes 1, 2) | 35   |      | 35                   |      | 35     |      | ns    |                   |
| TCLR1X  | RDY Hold Time into 8284A (See Notes 1, 2)  | 0    |      | 0                    |      | 0      |      | ns    |                   |
| TRYHCH  | READY Setup Time into 8086                 | 118  |      | 53                   |      | 68     |      | ns    |                   |
| TCHRYX  | READY Hold Time into 8086                  | 30   |      | 20                   |      | 20     |      | ns    |                   |
| TRYLCL  | READY Inactive to CLK (See Note 3)         | -8   |      | -10                  |      | -8     |      | ns    |                   |
| THVCH   | HOLD Setup Time                            | 35   |      | 20                   |      | 20     |      | ns    |                   |
| TINVCH  | INTR, NMI, TEST Setup Time (See Note 2)    | 30   |      | 15                   |      | 15     |      | ns    |                   |
| TILIH   | Input Rise Time (Except CLK)               |      | 20   |                      | 20   |        | 20   | ns    | From 0.8V to 2.0V |
| TIHIL   | Input Fall Time (Except CLK)               |      | 12   |                      | 12   |        | 12   | ns    | From 2.0V to 0.8V |

## A.C. CHARACTERISTICS (Continued)

## TIMING RESPONSES

| Symbol | Parameter                                       | 8086      |      | 8086-1 (Preliminary) |      | 8086-2    |      | Units | Test Conditions                                                                        |
|--------|-------------------------------------------------|-----------|------|----------------------|------|-----------|------|-------|----------------------------------------------------------------------------------------|
|        |                                                 | Min.      | Max. | Min.                 | Max. | Min.      | Max. |       |                                                                                        |
| TCLAV  | Address Valid Delay                             | 10        | 110  | 10                   | 50   | 10        | 60   | ns    | *C <sub>L</sub> = 20-100 pF<br>for all 8086 Outputs<br>(In addition to 8086 self-load) |
| TCLAX  | Address Hold Time                               | 10        |      | 10                   |      | 10        |      | ns    |                                                                                        |
| TCLAZ  | Address Float Delay                             | TCLAX     | 80   | 10                   | 40   | TCLAX     | 50   | ns    |                                                                                        |
| TLHLL  | ALE Width                                       | TCLCH-20  |      | TCLCH-10             |      | TCLCH-10  |      | ns    |                                                                                        |
| TCLLH  | ALE Active Delay                                |           | 80   |                      | 40   |           | 50   | ns    |                                                                                        |
| TCHLL  | ALE Inactive Delay                              |           | 85   |                      | 45   |           | 55   | ns    |                                                                                        |
| TLLAX  | Address Hold Time to ALE Inactive               | TCHCL-10  |      | TCHCL-10             |      | TCHCL-10  |      | ns    |                                                                                        |
| TCLDV  | Data Valid Delay                                | 10        | 110  | 10                   | 50   | 10        | 60   | ns    |                                                                                        |
| TCHDX  | Data Hold Time                                  | 10        |      | 10                   |      | 10        |      | ns    |                                                                                        |
| TWHDX  | Data Hold Time After WR                         | TCLCH-30  |      | TCLCH-25             |      | TCLCH-30  |      | ns    |                                                                                        |
| TCVCTV | Control Active Delay 1                          | 10        | 110  | 10                   | 50   | 10        | 70   | ns    |                                                                                        |
| TCHCTV | Control Active Delay 2                          | 10        | 110  | 10                   | 45   | 10        | 60   | ns    |                                                                                        |
| TCVCTX | Control Inactive Delay                          | 10        | 110  | 10                   | 50   | 10        | 70   | ns    |                                                                                        |
| TAZRL  | Address Float to READ Active                    | 0         |      | 0                    |      | 0         |      | ns    |                                                                                        |
| TCLRL  | $\overline{RD}$ Active Delay                    | 10        | 165  | 10                   | 70   | 10        | 100  | ns    |                                                                                        |
| TCLRH  | $\overline{RD}$ Inactive Delay                  | 10        | 150  | 10                   | 60   | 10        | 80   | ns    |                                                                                        |
| TRHAV  | $\overline{RD}$ Inactive to Next Address Active | TCLCL-45  |      | TCLCL-35             |      | TCLCL-40  |      | ns    |                                                                                        |
| TCLHAV | HLDA Valid Delay                                | 10        | 160  | 10                   | 60   | 10        | 100  | ns    |                                                                                        |
| TRLRH  | $\overline{RD}$ Width                           | 2TCLCL-75 |      | 2TCLCL-40            |      | 2TCLCL-50 |      | ns    |                                                                                        |
| TWLWH  | WR Width                                        | 2TCLCL-60 |      | 2TCLCL-35            |      | 2TCLCL-40 |      | ns    |                                                                                        |
| TAVAL  | Address Valid to ALE Low                        | TCLCH-60  |      | TCLCH-35             |      | TCLCH-40  |      | ns    |                                                                                        |
| TOLOH  | Output Rise Time                                |           | 20   |                      | 20   |           | 20   | ns    | From 0.8V to 2.0V                                                                      |
| TOHOL  | Output Fall Time                                |           | 12   |                      | 12   |           | 12   | ns    | From 2.0V to 0.8V                                                                      |

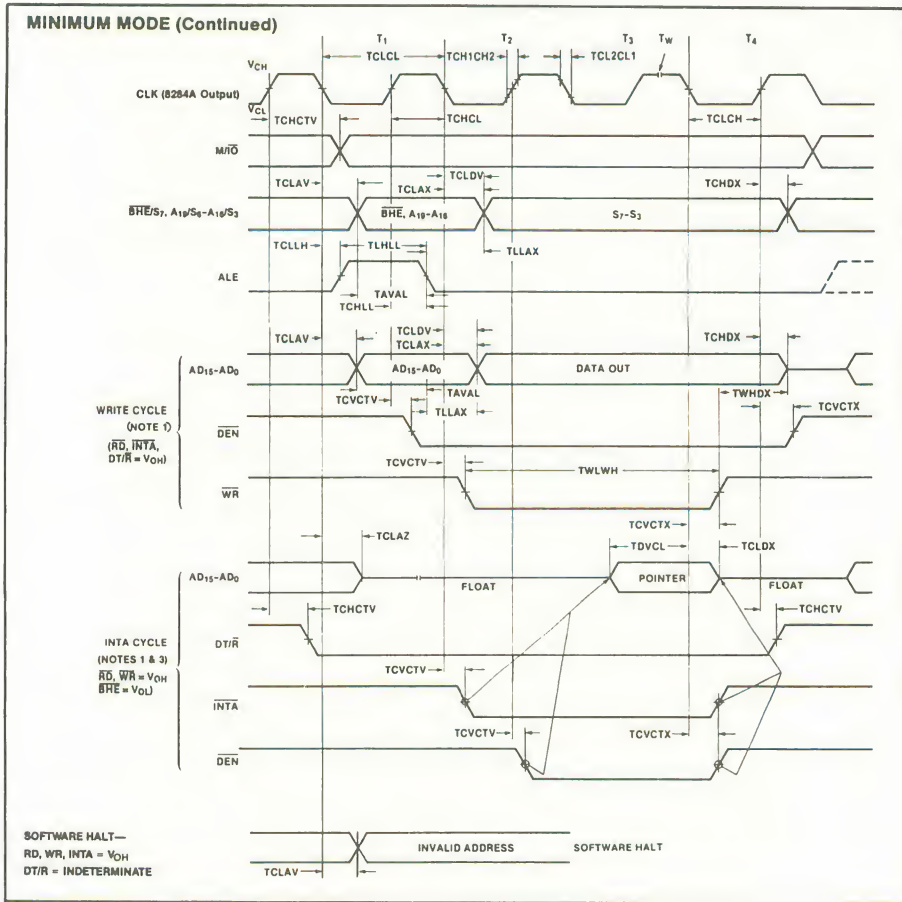
## NOTES:

- Signal at 8284A shown for reference only.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T2 state. (8 ns into T3).





## WAVEFORMS (Continued)



## NOTES:

1. All signals switch between V<sub>OH</sub> and V<sub>OL</sub> unless otherwise specified.
2. RDY is sampled near the end of T<sub>2</sub>, T<sub>3</sub>, T<sub>W</sub> to determine if T<sub>W</sub> machine states are to be inserted.
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 8284A are shown for reference only.
5. All timing measurements are made at 1.5V unless otherwise noted.

## A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)  
TIMING REQUIREMENTS

| Symbol  | Parameter                                                 | 8086 |      | 8086-1 (Preliminary) |      | 8086-2 (Preliminary) |      | Units | Test Conditions   |
|---------|-----------------------------------------------------------|------|------|----------------------|------|----------------------|------|-------|-------------------|
|         |                                                           | Min. | Max. | Min.                 | Max. | Min.                 | Max. |       |                   |
| TCLCL   | CLK Cycle Period                                          | 200  | 500  | 100                  | 500  | 125                  | 500  | ns    |                   |
| TCLCH   | CLK Low Time                                              | 118  |      | 53                   |      | 68                   |      | ns    |                   |
| TCHCL   | CLK High Time                                             | 69   |      | 39                   |      | 44                   |      | ns    |                   |
| TCH1CH2 | CLK Rise Time                                             |      | 10   |                      | 10   |                      | 10   | ns    | From 1.0V to 3.5V |
| TCL2CL1 | CLK Fall Time                                             |      | 10   |                      | 10   |                      | 10   | ns    | From 3.5V to 1.0V |
| TDVCL   | Data in Setup Time                                        | 30   |      | 5                    |      | 20                   |      | ns    |                   |
| TCLDX   | Data In Hold Time                                         | 10   |      | 10                   |      | 10                   |      | ns    |                   |
| TR1VCL  | RDY Setup Time into 8284A (See Notes 1, 2)                | 35   |      | 35                   |      | 35                   |      | ns    |                   |
| TCLR1X  | RDY Hold Time into 8284A (See Notes 1, 2)                 | 0    |      | 0                    |      | 0                    |      | ns    |                   |
| TRYHCH  | READY Setup Time into 8086                                | 118  |      | 53                   |      | 68                   |      | ns    |                   |
| TCHRYX  | READY Hold Time into 8086                                 | 30   |      | 20                   |      | 20                   |      | ns    |                   |
| TRYLCL  | READY Inactive to CLK (See Note 4)                        | -8   |      | -10                  |      | -8                   |      | ns    |                   |
| TINVCH  | Setup Time for Recognition (INTR, NMI, TEST) (See Note 2) | 30   |      | 15                   |      | 15                   |      | ns    |                   |
| TGVCH   | $\overline{RQ}/\overline{GT}$ Setup Time                  | 30   |      | 12                   |      | 15                   |      | ns    |                   |
| TCHGX   | $\overline{RQ}$ Hold Time into 8086                       | 40   |      | 20                   |      | 30                   |      | ns    |                   |
| TILIH   | Input Rise Time (Except CLK)                              |      | 20   |                      | 20   |                      | 20   | ns    | From 0.8V to 2.0V |
| TIHIL   | Input Fall Time (Except CLK)                              |      | 12   |                      | 12   |                      | 12   | ns    | From 2.0V to 0.8V |

## NOTES:

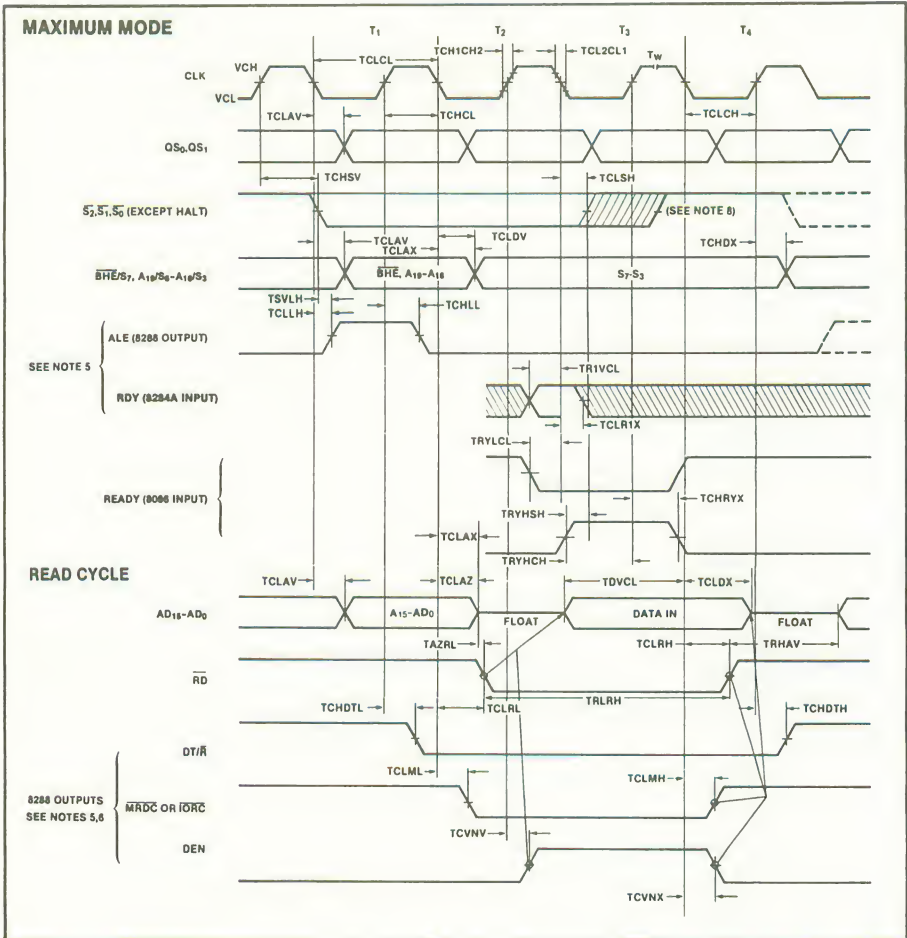
- Signal at 8284A or 8288 shown for reference only.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T3 and wait states.
- Applies only to T2 state (8 ns into T3).

## A.C. CHARACTERISTICS (Continued)

## TIMING RESPONSES

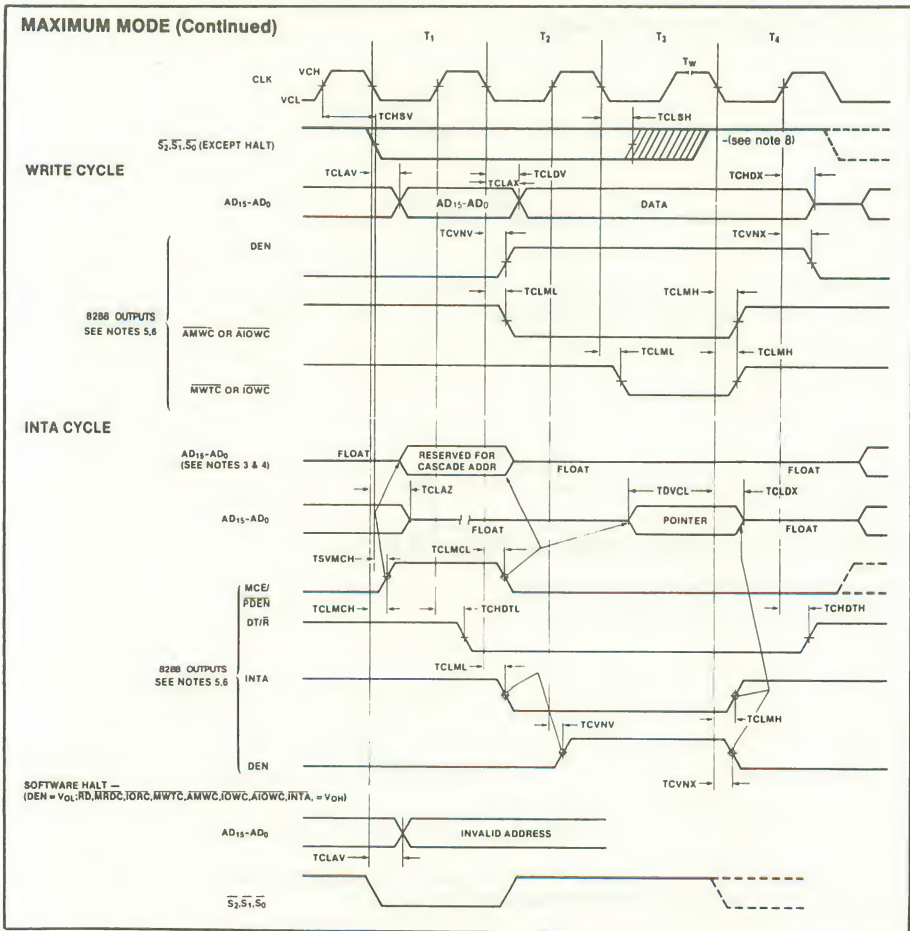
| Symbol | Parameter                                     | 8086      |      | 8086-1 (Preliminary) |      | 8086-2 (Preliminary) |      | Units | Test Conditions                                                                 |
|--------|-----------------------------------------------|-----------|------|----------------------|------|----------------------|------|-------|---------------------------------------------------------------------------------|
|        |                                               | Min.      | Max. | Min.                 | Max. | Min.                 | Max. |       |                                                                                 |
| TCLML  | Command Active Delay (See Note 1)             | 10        | 35   | 10                   | 35   | 10                   | 35   | ns    | C <sub>L</sub> = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load) |
| TCLMH  | Command Inactive Delay (See Note 1)           | 10        | 35   | 10                   | 35   | 10                   | 35   | ns    |                                                                                 |
| TRYHSH | READY Active to Status Passive (See Note 3)   |           | 110  |                      | 45   |                      | 65   | ns    |                                                                                 |
| TCHSV  | Status Active Delay                           | 10        | 110  | 10                   | 45   | 10                   | 60   | ns    |                                                                                 |
| TCLSH  | Status Inactive Delay                         | 10        | 130  | 10                   | 55   | 10                   | 70   | ns    |                                                                                 |
| TCLAV  | Address Valid Delay                           | 10        | 110  | 10                   | 50   | 10                   | 60   | ns    |                                                                                 |
| TCLAX  | Address Hold Time                             | 10        |      | 10                   |      | 10                   |      | ns    |                                                                                 |
| TCLAZ  | Address Float Delay                           | TCLAX     | 80   | 10                   | 40   | TCLAX                | 50   | ns    |                                                                                 |
| TSVLH  | Status Valid to ALE High (See Note 1)         |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TSVMCH | Status Valid to MCE High (See Note 1)         |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TCLLH  | CLK Low to ALE Valid (See Note 1)             |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TCLMCH | CLK Low to MCE High (See Note 1)              |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TCHLL  | ALE Inactive Delay (See Note 1)               |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TCLMCL | MCE Inactive Delay (See Note 1)               |           | 15   |                      | 15   |                      | 15   | ns    |                                                                                 |
| TCLDV  | Data Valid Delay                              | 10        | 110  | 10                   | 50   | 10                   | 60   | ns    |                                                                                 |
| TCHDX  | Data Hold Time                                | 10        |      | 10                   |      | 10                   |      | ns    |                                                                                 |
| TCVNV  | Control Active Delay (See Note 1)             | 5         | 45   | 5                    | 45   | 5                    | 45   | ns    |                                                                                 |
| TCVNX  | Control Inactive Delay (See Note 1)           | 10        | 45   | 10                   | 45   | 10                   | 45   | ns    |                                                                                 |
| TAZRL  | Address Float to Read Active                  | 0         |      | 0                    |      | 0                    |      | ns    |                                                                                 |
| TCLRL  | RD Active Delay                               | 10        | 165  | 10                   | 70   | 10                   | 100  | ns    |                                                                                 |
| TCLRH  | RD Inactive Delay                             | 10        | 150  | 10                   | 60   | 10                   | 80   | ns    |                                                                                 |
| TRHAV  | RD Inactive to Next Address Active            | TCLCL-45  |      | TCLCL-35             |      | TCLCL-40             |      | ns    |                                                                                 |
| TCHDTL | Direction Control Active Delay (See Note 1)   |           | 50   |                      | 50   |                      | 50   | ns    |                                                                                 |
| TCHDTH | Direction Control Inactive Delay (See Note 1) |           | 30   |                      | 30   |                      | 30   | ns    |                                                                                 |
| TCLGL  | GT Active Delay                               | 0         | 85   | 0                    | 45   | 0                    | 50   | ns    |                                                                                 |
| TCLGH  | GT Inactive Delay                             | 0         | 85   | 0                    | 45   | 0                    | 50   | ns    |                                                                                 |
| TRLRH  | RD Width                                      | 2TCLCL-75 |      | 2TCLCL-40            |      | 2TCLCL-50            |      | ns    |                                                                                 |
| TOLOH  | Output Rise Time                              |           | 20   |                      | 20   |                      | 20   | ns    | From 0.8V to 2.0V                                                               |
| TOHOL  | Output Fall Time                              |           | 12   |                      | 12   |                      | 12   | ns    | From 2.0V to 0.8V                                                               |

## WAVEFORMS





## WAVEFORMS (Continued)

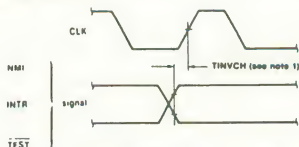


**NOTES:**

1. All signals switch between  $V_{OH}$  and  $V_{OL}$  unless otherwise specified.
2. RDY is sampled near the end of  $T_2$ .  $T_2$ ,  $T_3$  to determine if  $T_W$  machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 8284A or 8288 are shown for reference only.
6. The issuance of the 8288 command and control signals ( $\overline{MRDC}$ ,  $\overline{MWTC}$ ,  $\overline{AMWC}$ ,  $\overline{IORC}$ ,  $\overline{IOWC}$ ,  $\overline{AIOWC}$ , INTA and DEN) lags the active high 8288 CEN.
7. All timing measurements are made at 1.5V unless otherwise noted.
8. Status inactive in state just prior to  $T_4$ .

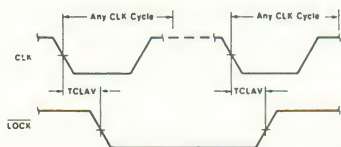
## WAVEFORMS (Continued)

## ASYNCHRONOUS SIGNAL RECOGNITION

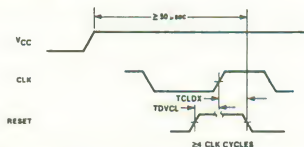


NOTE: 1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

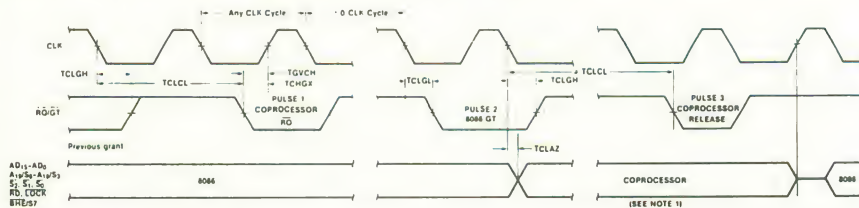
## BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



## RESET TIMING

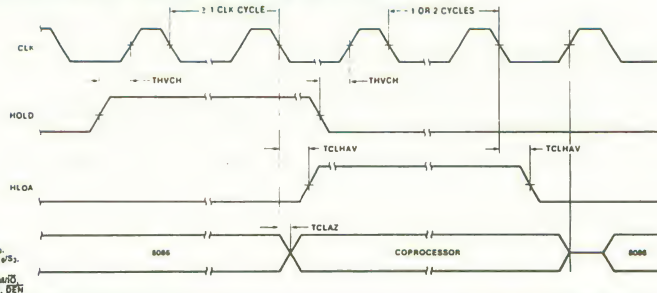


## REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTES 1 THE COPROCESSOR MAY NOT DRIVE THE BUSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION

## HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



## iAPX 88/10 8-BIT HMOS MICROPROCESSOR 8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with iAPX 86/10 (8086 CPU)
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Compatible with 8155-2, 8755A-2 and 8185-2 Multiplexed Peripherals
- Two Clock Rates:  
5 MHz for 8088  
8 MHz for 8088-2

The Intel® iAPX 88/10 is a new generation, high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with iAPX 86/10 software and 8080/8085 hardware and peripherals.

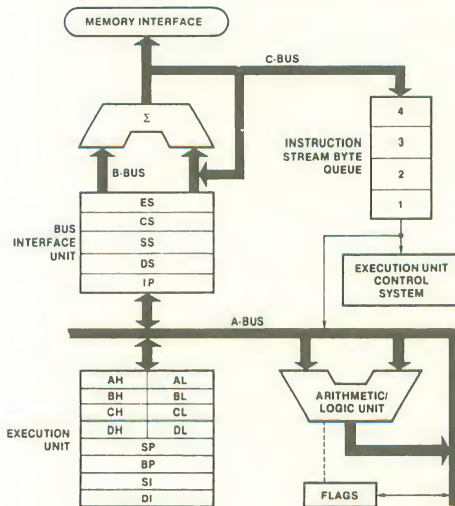


Figure 1. iAPX 88/10 CPU Functional Block Diagram

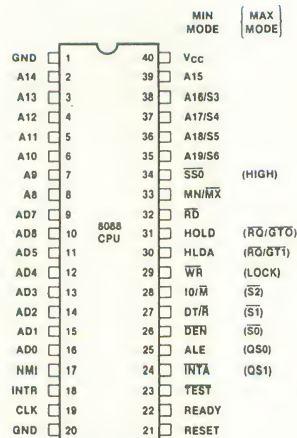


Figure 2. iAPX 88/10 Pin Configuration

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . . . 0°C to 70°C  
 Storage Temperature . . . . . - 65°C to + 150°C  
 Voltage on Any Pin with  
   Respect to Ground . . . . . - 1.0 to + 7V  
 Power Dissipation . . . . . 2.5 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**D.C. CHARACTERISTICS**

(8088:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 10\%$ )  
 (8088-2:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )

| Symbol   | Parameter                                                                                     | Min. | Max.           | Units         | Test Conditions                         |
|----------|-----------------------------------------------------------------------------------------------|------|----------------|---------------|-----------------------------------------|
| $V_{IL}$ | Input Low Voltage                                                                             | -0.5 | +0.8           | V             |                                         |
| $V_{IH}$ | Input High Voltage                                                                            | 2.0  | $V_{CC} + 0.5$ | V             |                                         |
| $V_{OL}$ | Output Low Voltage                                                                            |      | 0.45           | V             | $I_{OL} = 2.0\text{ mA}$                |
| $V_{OH}$ | Output High Voltage                                                                           | 2.4  |                | V             | $I_{OH} = -400\text{ }\mu\text{A}$      |
| $I_{CC}$ | Power Supply Current: 8088<br>8088-2                                                          |      | 340<br>350     | mA            | $T_A = 25^\circ\text{C}$                |
| $I_{LI}$ | Input Leakage Current                                                                         |      | $\pm 10$       | $\mu\text{A}$ | $0\text{V} \leq V_{IN} \leq V_{CC}$     |
| $I_{LO}$ | Output Leakage Current                                                                        |      | $\pm 10$       | $\mu\text{A}$ | $0.45\text{V} \leq V_{OUT} \leq V_{CC}$ |
| $V_{CL}$ | Clock Input Low Voltage                                                                       | -0.5 | +0.6           | V             |                                         |
| $V_{CH}$ | Clock Input High Voltage                                                                      | 3.9  | $V_{CC} + 1.0$ | V             |                                         |
| $C_{IN}$ | Capacitance if Input Buffer<br>(All input except<br>AD <sub>0</sub> -AD <sub>7</sub> , RQ/GT) |      | 15             | pF            | $f_c = 1\text{ MHz}$                    |
| $C_{IO}$ | Capacitance of I/O Buffer<br>(AD <sub>0</sub> -AD <sub>7</sub> , RQ/GT)                       |      | 15             | pF            | $f_c = 1\text{ MHz}$                    |

**A.C. CHARACTERISTICS** (8088:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 10\%$ )  
 (8088-2:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ )

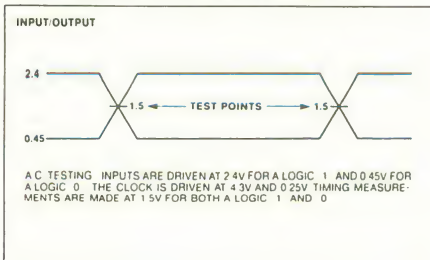
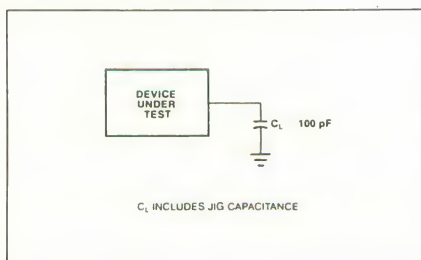
**MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS**

| Symbol  | Parameter                                                   | 8088                               |      | 8088-2                             |      | Units | Test Conditions   |
|---------|-------------------------------------------------------------|------------------------------------|------|------------------------------------|------|-------|-------------------|
|         |                                                             | Min.                               | Max. | Min.                               | Max. |       |                   |
| TCLCL   | CLK Cycle Period                                            | 200                                | 500  | 125                                | 500  | ns    |                   |
| TCLCH   | CLK Low Time                                                | $(\frac{2}{3} \text{ TCLCL}) - 15$ |      | $(\frac{2}{3} \text{ TCLCL}) - 15$ |      | ns    |                   |
| TCHCL   | CLK High Time                                               | $(\frac{1}{3} \text{ TCLCL}) + 2$  |      | $(\frac{1}{3} \text{ TCLCL}) + 2$  |      | ns    |                   |
| TCH1CH2 | CLK Rise Time                                               |                                    | 10   |                                    | 10   | ns    | From 1.0V to 3.5V |
| TCL2CL1 | CLK Fall Time                                               |                                    | 10   |                                    | 10   | ns    | From 3.5V to 1.0V |
| TDVCL   | Data in Setup Time                                          | 30                                 |      | 20                                 |      | ns    |                   |
| TCLDX   | Data in Hold Time                                           | 10                                 |      | 10                                 |      | ns    |                   |
| TR1VCL  | RDY Setup Time into 8284 (See Notes 1, 2)                   | 35                                 |      | 35                                 |      | ns    |                   |
| TCLR1X  | RDY Hold Time into 8284 (See Notes 1, 2)                    | 0                                  |      | 0                                  |      | ns    |                   |
| TRYHCH  | READY Setup Time into 8088                                  | $(\frac{2}{3} \text{ TCLCL}) - 15$ |      | $(\frac{2}{3} \text{ TCLCL}) - 15$ |      | ns    |                   |
| TCHRYX  | READY Hold Time into 8088                                   | 30                                 |      | 20                                 |      | ns    |                   |
| TRYLCL  | READY Inactive to CLK (See Note 3)                          | -8                                 |      | -8                                 |      | ns    |                   |
| THVCH   | HOLD Setup Time                                             | 35                                 |      | 20                                 |      | ns    |                   |
| TINVCH  | INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2) | 30                                 |      | 15                                 |      | ns    |                   |
| TILIH   | Input Rise Time (Except CLK)                                |                                    | 20   |                                    | 20   | ns    | From 0.8V to 2.0V |
| TIHIL   | Input Fall Time (Except CLK)                                |                                    | 12   |                                    | 12   | ns    | From 2.0V to 0.8V |



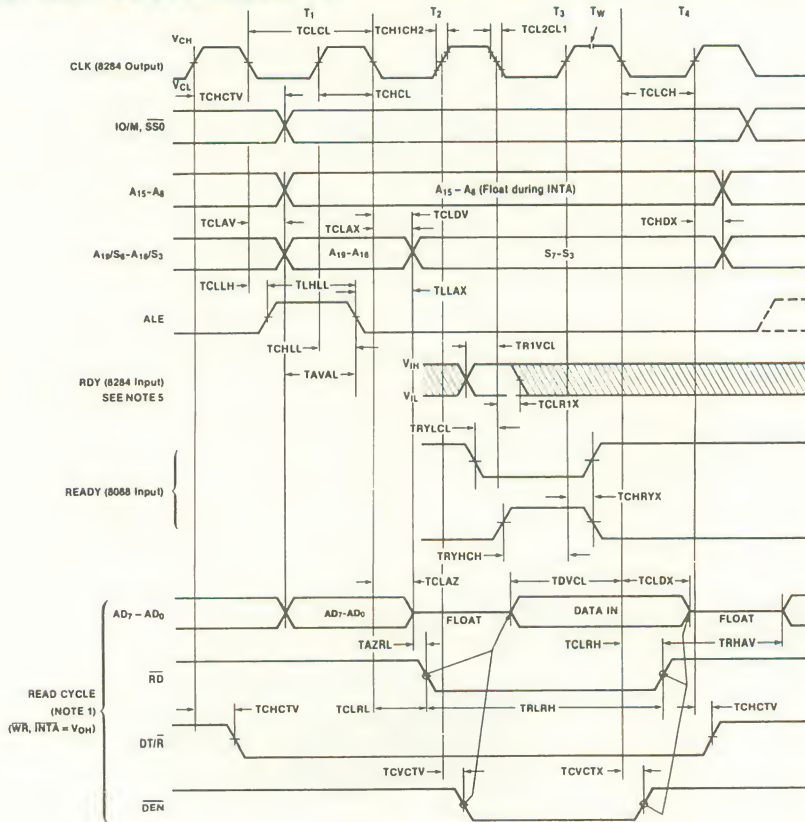
**A.C. CHARACTERISTICS (Continued)****TIMING RESPONSES**

| Symbol | Parameter                                       | 8088      |      | 8088-2    |      | Units | Test Conditions                                                                    |
|--------|-------------------------------------------------|-----------|------|-----------|------|-------|------------------------------------------------------------------------------------|
|        |                                                 | Min.      | Max. | Min.      | Max. |       |                                                                                    |
| TCLAV  | Address Valid Delay                             | 10        | 110  | 10        | 60   | ns    | $C_L = 20\text{-}100\text{ pF}$ for all 8088 Outputs in addition to internal loads |
| TCLAX  | Address Hold Time                               | 10        |      | 10        |      | ns    |                                                                                    |
| TCLAZ  | Address Float Delay                             | TCLAX     | 80   | TCLAX     | 50   | ns    |                                                                                    |
| TLHLZ  | ALE Width                                       | TCLCH-20  |      | TCLCH-10  |      | ns    |                                                                                    |
| TCLLH  | ALE Active Delay                                |           | 80   |           | 50   | ns    |                                                                                    |
| TCHLL  | ALE Inactive Delay                              |           | 85   |           | 55   | ns    |                                                                                    |
| TLLAX  | Address Hold Time to ALE Inactive               | TCHCL-10  |      | TCHCL-10  |      | ns    |                                                                                    |
| TCLDV  | Data Valid Delay                                | 10        | 110  | 10        | 60   | ns    |                                                                                    |
| TCHDX  | Data Hold Time                                  | 10        |      | 10        |      | ns    |                                                                                    |
| TWHDX  | Data Hold Time After $\overline{WR}$            | TCLCH-30  |      | TCLCH-30  |      | ns    |                                                                                    |
| TCVCTV | Control Active Delay 1                          | 10        | 110  | 10        | 70   | ns    |                                                                                    |
| TCHCTV | Control Active Delay 2                          | 10        | 110  | 10        | 60   | ns    |                                                                                    |
| TCVCTX | Control Inactive Delay                          | 10        | 110  | 10        | 70   | ns    |                                                                                    |
| TAZRL  | Address Float to READ Active                    | 0         |      | 0         |      | ns    |                                                                                    |
| TCLRL  | $\overline{RD}$ Active Delay                    | 10        | 165  | 10        | 100  | ns    |                                                                                    |
| TCLRH  | $\overline{RD}$ Inactive Delay                  | 10        | 150  | 10        | 80   | ns    |                                                                                    |
| TRHAV  | $\overline{RD}$ Inactive to Next Address Active | TCLCL-45  |      | TCLCL-40  |      | ns    | From 0.8V to 2.0V                                                                  |
| TCLHAV | HLDA Valid Delay                                | 10        | 160  | 10        | 100  | ns    |                                                                                    |
| TRLRH  | $\overline{RD}$ Width                           | 2TCLCL-75 |      | 2TCLCL-50 |      | ns    |                                                                                    |
| TWLWH  | $\overline{WR}$ Width                           | 2TCLCL-60 |      | 2TCLCL-40 |      | ns    |                                                                                    |
| TAVAL  | Address Valid to ALE Low                        | TCLCH-60  |      | TCLCH-40  |      | ns    |                                                                                    |
| TOLOH  | Output Rise Time                                |           | 20   |           | 20   | ns    | From 2.0V to 0.8V                                                                  |
| TOHOL  | Output Fall Time                                |           | 12   |           | 12   | ns    |                                                                                    |

**A.C. TESTING INPUT, OUTPUT WAVEFORM****A.C. TESTING LOAD CIRCUIT**

## WAVEFORMS

### BUS TIMING—MINIMUM MODE SYSTEM





**A.C. CHARACTERISTICS****MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)****TIMING REQUIREMENTS**

| Symbol  | Parameter                                                       | 8088                               |      | 8088-2                             |      | Units | Test Conditions   |
|---------|-----------------------------------------------------------------|------------------------------------|------|------------------------------------|------|-------|-------------------|
|         |                                                                 | Min.                               | Max. | Min.                               | Max. |       |                   |
| TCLCL   | CLK Cycle Period                                                | 200                                | 500  | 125                                | 500  | ns    |                   |
| TCLCH   | CLK Low Time                                                    | $(\frac{1}{2} \text{ TCLCL}) - 15$ |      | $(\frac{1}{2} \text{ TCLCL}) - 15$ |      | ns    |                   |
| TCHCL   | CLK High Time                                                   | $(\frac{1}{2} \text{ TCLCL}) + 2$  |      | $(\frac{1}{2} \text{ TCLCL}) + 2$  |      | ns    |                   |
| TCH1CH2 | CLK Rise Time                                                   |                                    | 10   |                                    | 10   | ns    | From 1.0V to 3.5V |
| TCL2CL1 | CLK Fall Time                                                   |                                    | 10   |                                    | 10   | ns    | From 3.5V to 1.0V |
| TDVCL   | Data In Setup Time                                              | 30                                 |      | 20                                 |      | ns    |                   |
| TCLDX   | Data In Hold Time                                               | 10                                 |      | 10                                 |      | ns    |                   |
| TR1VCL  | RDY Setup Time into 8284<br>(See Notes 1, 2)                    | 35                                 |      | 35                                 |      | ns    |                   |
| TCLR1X  | RDY Hold Time into 8284<br>(See Notes 1, 2)                     | 0                                  |      | 0                                  |      | ns    |                   |
| TRYHCH  | READY Setup Time into 8088                                      | $(\frac{1}{2} \text{ TCLCL}) - 15$ |      | $(\frac{1}{2} \text{ TCLCL}) - 15$ |      | ns    |                   |
| TCHRYX  | READY Hold Time into 8088                                       | 30                                 |      | 20                                 |      | ns    |                   |
| TRYLCL  | READY Inactive to CLK (See Note 4)                              | -8                                 |      | -8                                 |      | ns    |                   |
| TINVCH  | Setup Time for Recognition<br>(INTR, NMI, TEST)<br>(See Note 2) | 30                                 |      | 15                                 |      | ns    |                   |
| TGVCH   | RQ/GT Setup Time                                                | 30                                 |      | 15                                 |      | ns    |                   |
| TCHGX   | RQ Hold Time into 8086                                          | 40                                 |      | 30                                 |      | ns    |                   |
| TILIH   | Input Rise Time<br>(Except CLK)                                 |                                    | 20   |                                    | 20   | ns    | From 0.8V to 2.0V |
| TIHIL   | Input Fall Time (Except CLK)                                    |                                    | 12   |                                    | 12   | ns    | From 2.0V to 0.8V |

**NOTES:**

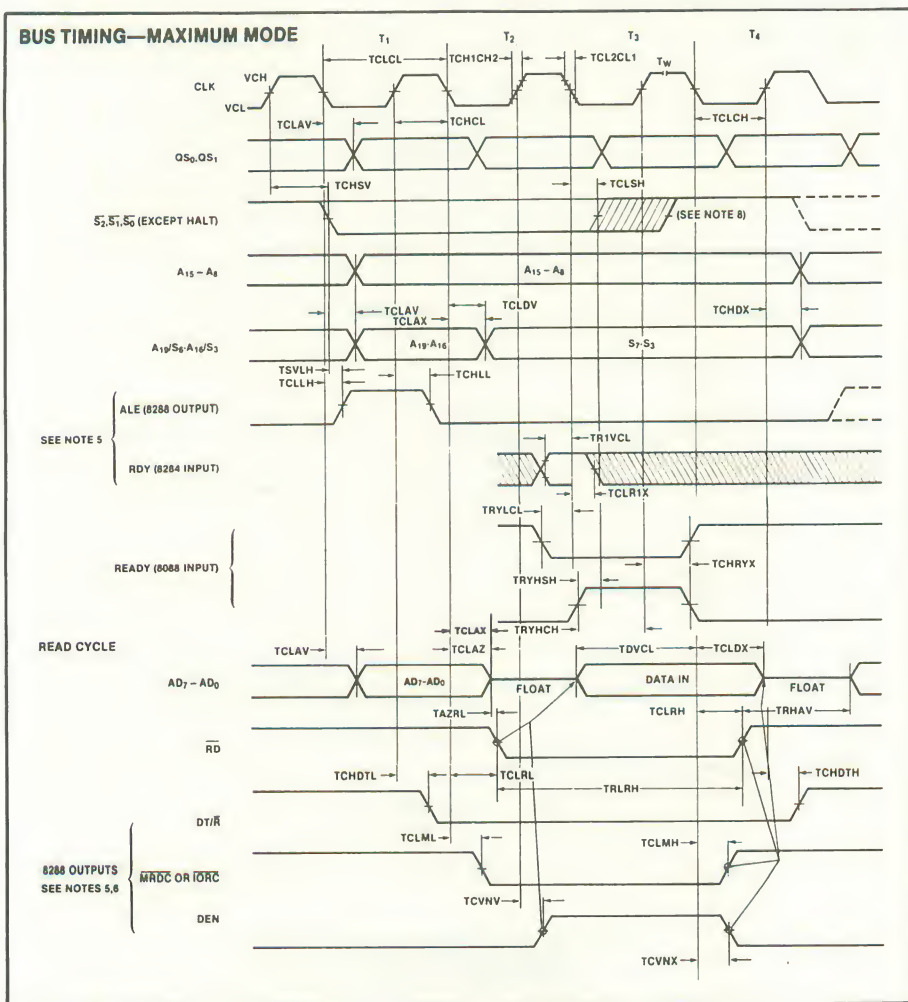
- Signal at 8284 or 8288 shown for reference only.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T2 state (8 ns into T3 state).
- Applies only to T2 state (8 ns into T3 state).

**A.C. CHARACTERISTICS****TIMING RESPONSES**

| Symbol | Parameter                                     | 8088      |      | 8088-2    |      | Units | Test Conditions                                                               |
|--------|-----------------------------------------------|-----------|------|-----------|------|-------|-------------------------------------------------------------------------------|
|        |                                               | Min.      | Max. | Min.      | Max. |       |                                                                               |
| TCLML  | Command Active Delay (See Note 1)             | 10        | 35   | 10        | 35   | ns    | C <sub>L</sub> = 20-100 pF for all 8088 Outputs in addition to internal loads |
| TCLMH  | Command Inactive Delay (See Note 1)           | 10        | 35   | 10        | 35   | ns    |                                                                               |
| TRYHSH | READY Active to Status Passive (See Note 3)   |           | 110  |           | 65   | ns    |                                                                               |
| TCHSV  | Status Active Delay                           | 10        | 110  | 10        | 60   | ns    |                                                                               |
| TCLSH  | Status Inactive Delay                         | 10        | 130  | 10        | 70   | ns    |                                                                               |
| TCLAV  | Address Valid Delay                           | 10        | 110  | 10        | 60   | ns    |                                                                               |
| TCLAX  | Address Hold Time                             | 10        |      | 10        |      | ns    |                                                                               |
| TCLAZ  | Address Float Delay                           | TCLAX     | 80   | TCLAX     | 50   | ns    |                                                                               |
| TSVLH  | Status Valid to ALE High (See Note 1)         |           | 15   |           | 15   | ns    |                                                                               |
| TSVMCH | Status Valid to MCE High (See Note 1)         |           | 15   |           | 15   | ns    |                                                                               |
| TCLLH  | CLK Low to ALE Valid (See Note 1)             |           | 15   |           | 15   | ns    |                                                                               |
| TCLMCH | CLK Low to MCE High (See Note 1)              |           | 15   |           | 15   | ns    |                                                                               |
| TCHLL  | ALE Inactive Delay (See Note 1)               |           | 15   |           | 15   | ns    |                                                                               |
| TCLMCL | MCE Inactive Delay (See Note 1)               |           | 15   |           | 15   | ns    |                                                                               |
| TCLDV  | Data Valid Delay                              | 10        | 110  | 10        | 60   | ns    |                                                                               |
| TCHDX  | Data Hold Time                                | 10        |      | 10        |      | ns    |                                                                               |
| TCVNV  | Control Active Delay (See Note 1)             | 5         | 45   | 5         | 45   | ns    |                                                                               |
| TCVNX  | Control Inactive Delay (See Note 1)           | 10        | 45   | 10        | 45   | ns    |                                                                               |
| TAZRL  | Address Float to Read Active                  | 0         |      | 0         |      | ns    |                                                                               |
| TCLRL  | RD Active Delay                               | 10        | 165  | 10        | 100  | ns    |                                                                               |
| TCLRH  | RD Inactive Delay                             | 10        | 150  | 10        | 80   | ns    |                                                                               |
| TRHAV  | RD Inactive to Next Address Active            | TCLCL-45  |      | TCLCL-40  |      | ns    |                                                                               |
| TCHDTL | Direction Control Active Delay (See Note 1)   |           | 50   |           | 50   | ns    |                                                                               |
| TCHDTH | Direction Control Inactive Delay (See Note 1) |           | 30   |           | 30   | ns    |                                                                               |
| TCLGL  | GT Active Delay                               |           | 110  |           | 50   | ns    |                                                                               |
| TCLGH  | GT Inactive Delay                             |           | 85   |           | 50   | ns    |                                                                               |
| TRLRH  | RD Width                                      | 2TCLCL-75 |      | 2TCLCL-50 |      | ns    |                                                                               |
| TOLOH  | Output Rise Time                              |           | 20   |           | 20   | ns    |                                                                               |
| TOHOL  | Output Fall Time                              |           | 12   |           | 12   | ns    |                                                                               |



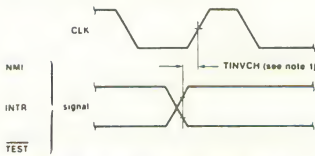
## WAVEFORMS





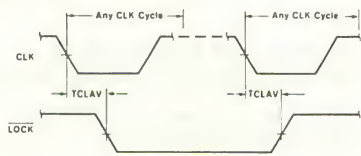
### WAVEFORMS (Continued)

## ASYNCHRONOUS SIGNAL RECOGNITION

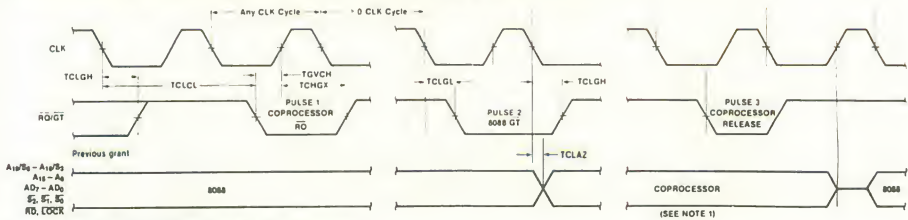


**NOTE: 1. SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK**

### BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)

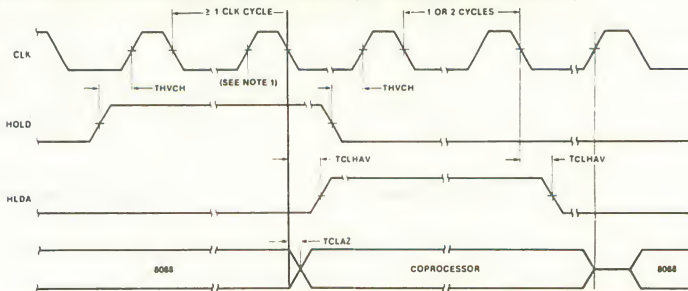


### REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTE: 1. THE COPROCESSOR MAY NOT DRIVE THE BUSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION.

### HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



## 8282/8283 OCTAL LATCH

- Address Latch for iAPX 86, 88, MCS-80®, MCS-85®, MCS-48® Families
- High Output Drive Capability for Driving System Data Bus
- Fully Parallel 8-Bit Data Register and Buffer
- Transparent during Active Strobe
- 3-State Outputs
- 20-Pin Package with 0.3" Center
- No Output Low Noise when Entering or Leaving High Impedance State

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

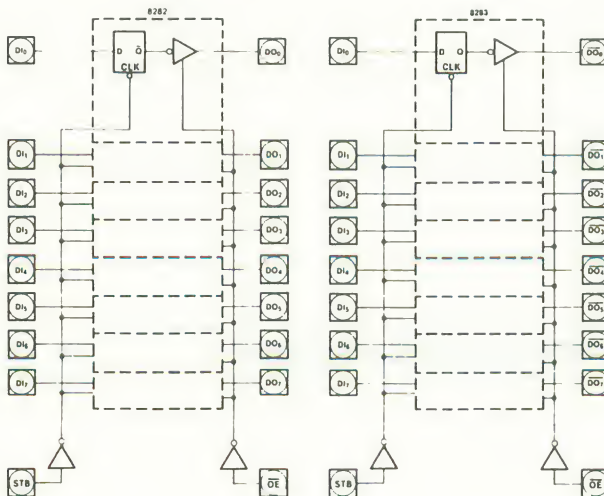


Figure 1. Logic Diagrams

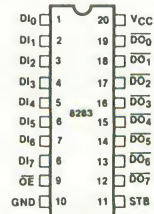
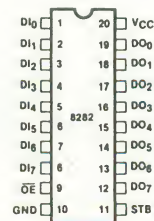


Figure 2. Pin Configurations

## ABSOLUTE MAXIMUM RATINGS\*

Temperature Under Bias.....0°C to 70°C  
 Storage Temperature.....-65°C to +150°C  
 All Output and Supply Voltages.....-0.5V to +7V  
 All Input Voltages.....-1.0V to +5.5V  
 Power Dissipation.....1 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## D.C. CHARACTERISTICS (V<sub>CC</sub> = 5V ±10%, T<sub>A</sub> = 0°C to 70°C)

| Symbol           | Parameter             | Min. | Max. | Units | Test Conditions                                                                      |
|------------------|-----------------------|------|------|-------|--------------------------------------------------------------------------------------|
| V <sub>C</sub>   | Input Clamp Voltage   |      | -1   | V     | I <sub>C</sub> = -5 mA                                                               |
| I <sub>CC</sub>  | Power Supply Current  |      | 160  | mA    |                                                                                      |
| I <sub>F</sub>   | Forward Input Current |      | -0.2 | mA    | V <sub>F</sub> = 0.45V                                                               |
| I <sub>R</sub>   | Reverse Input Current |      | 50   | μA    | V <sub>R</sub> = 5.25V                                                               |
| V <sub>OL</sub>  | Output Low Voltage    |      | .45  | V     | I <sub>OL</sub> = 32 mA                                                              |
| V <sub>OH</sub>  | Output High Voltage   | 2.4  |      | V     | I <sub>OH</sub> = -5 mA                                                              |
| I <sub>OFF</sub> | Output Off Current    |      | ± 50 | μA    | V <sub>OFF</sub> = 0.45 to 5.25V                                                     |
| V <sub>IL</sub>  | Input Low Voltage     |      | 0.8  | V     | V <sub>CC</sub> = 5.0V See Note 1                                                    |
| V <sub>IH</sub>  | Input High Voltage    | 2.0  |      | V     | V <sub>CC</sub> = 5.0V See Note 1                                                    |
| C <sub>IN</sub>  | Input Capacitance     |      | 12   | pF    | F = 1 MHz<br>V <sub>BIAS</sub> = 2.5V, V <sub>CC</sub> = 5V<br>T <sub>A</sub> = 25°C |

### NOTE:

1. Output Loading I<sub>OL</sub> = 32 mA, I<sub>OH</sub> = -5 mA, C<sub>L</sub> = 300 pF\*

## A.C. CHARACTERISTICS (V<sub>CC</sub> = 5V ±10%, T<sub>A</sub> = 0°C to 70°C Loading: Outputs—I<sub>OL</sub> = 32 mA, I<sub>OH</sub> = -5 mA, C<sub>L</sub> = 300 pF\*)

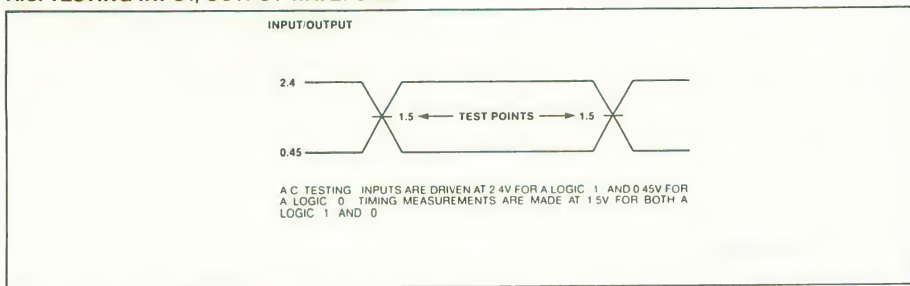
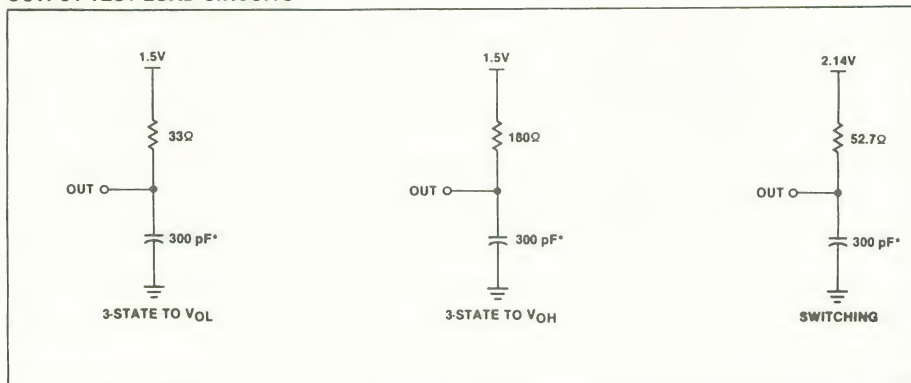
| Symbol                                | Parameter                                             | Min.     | Max.     | Units    | Test Conditions   |
|---------------------------------------|-------------------------------------------------------|----------|----------|----------|-------------------|
| T <sub>IOV</sub>                      | Input to Output Delay<br>—Inverting<br>—Non-Inverting | 5<br>5   | 22<br>30 | ns<br>ns | (See Note 1)      |
| T <sub>SHOV</sub>                     | STB to Output Delay<br>—Inverting<br>—Non-Inverting   | 10<br>10 | 40<br>45 | ns<br>ns |                   |
| T <sub>EH0Z</sub>                     | Output Disable Time                                   | 5        | 18       | ns       |                   |
| T <sub>ELOV</sub>                     | Output Enable Time                                    | 10       | 30       | ns       |                   |
| T <sub>IVSL</sub>                     | Input to STB Setup Time                               | 0        |          | ns       |                   |
| T <sub>SLIX</sub>                     | Input to STB Hold Time                                | 25       |          | ns       |                   |
| T <sub>SHSL</sub>                     | STB High Time                                         | 15       |          | ns       |                   |
| T <sub>ILIH</sub> , T <sub>OLOH</sub> | Input, Output Rise Time                               |          | 20       | ns       | From 0.8V to 2.0V |
| T <sub>IHIL</sub> , T <sub>OHOL</sub> | Input, Output Fall Time                               |          | 12       | ns       | From 2.0V to 0.8V |

### NOTE:

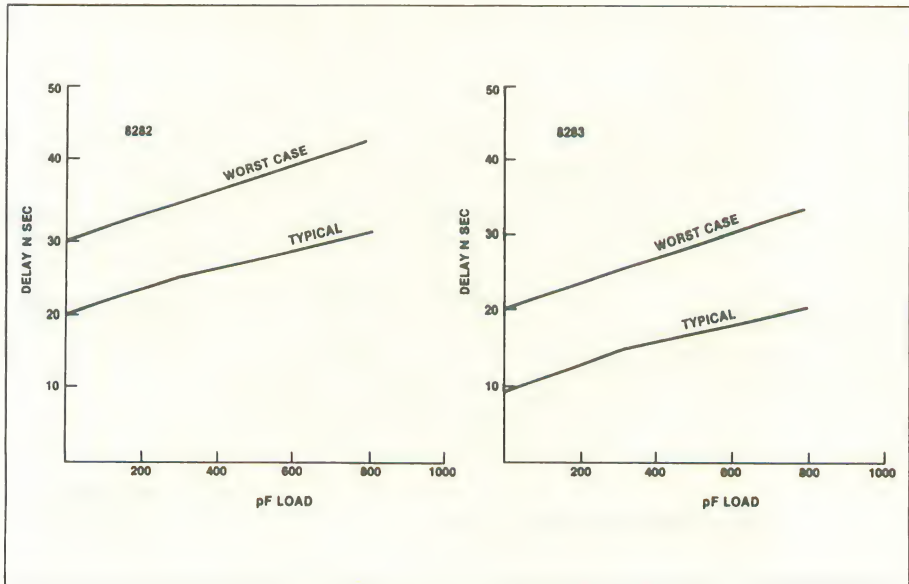
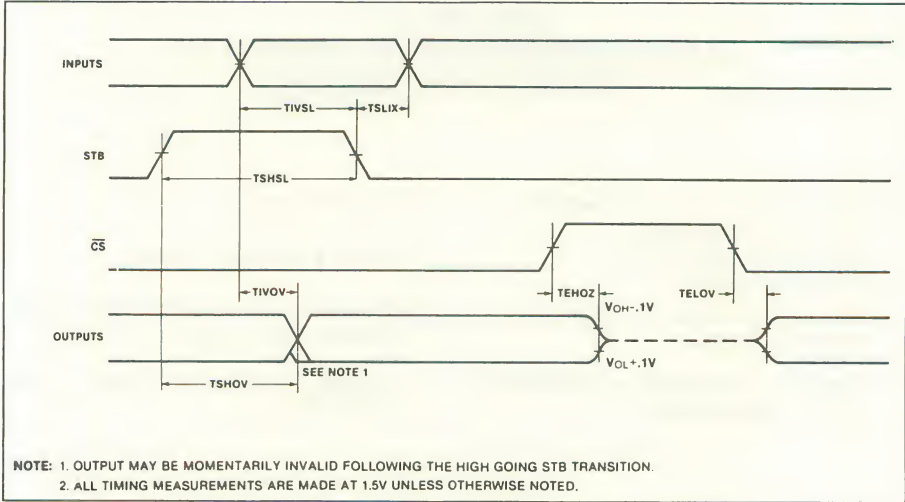
1. See waveforms and test load circuit on following page.

\*C<sub>L</sub> = 200 pF for plastic 8282/8283.



**A.C. TESTING INPUT, OUTPUT WAVEFORM****OUTPUT TEST LOAD CIRCUITS**

## WAVEFORMS



Output Delay vs. Capacitance

## 8284A/8284A-1 CLOCK GENERATOR AND DRIVER FOR iAPX 86, 88 PROCESSORS

- Generates the System Clock for the iAPX 86, 88 Processors:  
5 MHz, 8 MHz with 8284A  
10 MHz with 8284A-1
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and Multibus™ READY Synchronization
- 18-Pin Package
- Single +5V Power Supply
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other 8284As

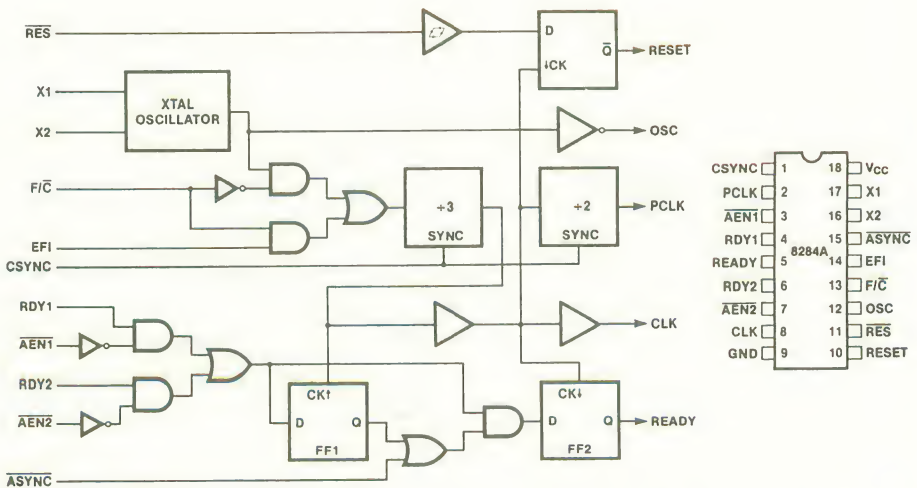
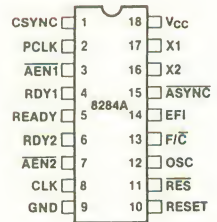


Figure 1. 8284A/8284A-1 Block Diagram

Figure 2. 8284A/8284A-1 Pin Configuration



**ABSOLUTE MAXIMUM RATINGS\***

Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 All Output and Supply Voltages ..... -0.5V to +7V  
 All Input Voltages ..... -1.0V to +5.5V  
 Power Dissipation ..... 1 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )

| Symbol              | Parameter                         | Min. | Max. | Units   | Test Conditions |
|---------------------|-----------------------------------|------|------|---------|-----------------|
| $I_F$               | Forward Input Current (ASYNC)     |      | -1.3 | mA      | $V_F = 0.45V$   |
|                     | Other Inputs                      |      | -0.5 | mA      | $V_F = 0.45V$   |
| $I_R$               | Reverse Input Current (ASYNC)     |      | 50   | $\mu A$ | $V_R = V_{CC}$  |
|                     | Other Inputs                      |      | 50   | $\mu A$ | $V_R = 5.25V$   |
| $V_C$               | Input Forward Clamp Voltage       |      | -1.0 | V       | $I_C = -5mA$    |
| $I_{CC}$            | Power Supply Current              |      | 162  | mA      |                 |
| $V_{IL}$            | Input LOW Voltage                 |      | 0.8  | V       |                 |
| $V_{IH}$            | Input HIGH Voltage                | 2.0  |      | V       |                 |
| $V_{IHR}$           | Reset Input HIGH Voltage          | 2.6  |      | V       |                 |
| $V_{OL}$            | Output LOW Voltage                |      | 0.45 | V       | 5mA             |
| $V_{OH}$            | Output HIGH Voltage CLK           | 4    |      | V       | -1mA            |
|                     | Other Outputs                     | 2.4  |      | V       | -1mA            |
| $V_{IHR} - V_{ILR}$ | $\overline{RES}$ Input Hysteresis | 0.25 |      | V       |                 |

**A.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )**TIMING REQUIREMENTS**

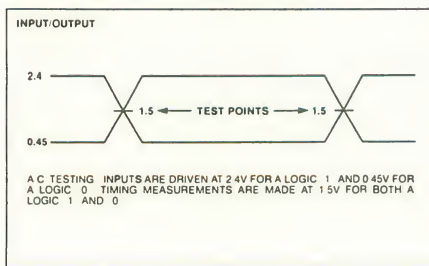
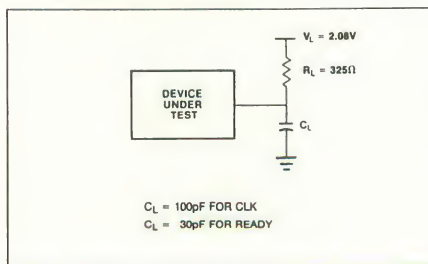
| Symbol       | Parameter                                                 | Min.              | Max. | Units | Test Conditions                  |
|--------------|-----------------------------------------------------------|-------------------|------|-------|----------------------------------|
| $t_{EHL}$    | External Frequency HIGH Time                              | 13                |      | ns    | 90% - 90% $V_{IN}$               |
| $t_{ELEH}$   | External Frequency LOW Time                               | 13                |      | ns    | 10% - 10% $V_{IN}$               |
| $t_{EEL}$    | EFI Period                                                | 33                |      | ns    | (Note 1)                         |
|              | XTAL Frequency                                            | 12                | 25   | MHz   |                                  |
| $t_{R1VCL}$  | RDY1, RDY2 Active Setup to CLK                            | 35                |      | ns    | $\overline{ASYNC} = \text{HIGH}$ |
| $t_{R1VCH}$  | RDY1, RDY2 Active Setup to CLK                            | 35                |      | ns    | $\overline{ASYNC} = \text{LOW}$  |
| $t_{R1VCL}$  | RDY1, RDY2 Inactive Setup to CLK                          | 35                |      | ns    |                                  |
| $t_{CLR1X}$  | RDY1, RDY2 Hold to CLK                                    | 0                 |      | ns    |                                  |
| $t_{AYVCL}$  | $\overline{ASYNC}$ Setup to CLK                           | 50                |      | ns    |                                  |
| $t_{CLAYX}$  | $\overline{ASYNC}$ Hold to CLK                            | 0                 |      | ns    |                                  |
| $t_{A1VR1V}$ | $\overline{AEN1}$ , $\overline{AEN2}$ Setup to RDY1, RDY2 | 15                |      | ns    |                                  |
| $t_{CLA1X}$  | $\overline{AEN1}$ , $\overline{AEN2}$ Hold to CLK         | 0                 |      | ns    |                                  |
| $t_{YHEH}$   | CSYNC Setup to EFI                                        | 20                |      | ns    |                                  |
| $t_{EHYL}$   | CSYNC Hold to EFI                                         | 10                |      | ns    |                                  |
| $t_{YHYL}$   | CSYNC Width                                               | $2 \cdot t_{EEL}$ |      | ns    |                                  |
| $t_{I1HCL}$  | $\overline{RES}$ Setup to CLK                             | 65                |      | ns    | (Note 1)                         |
| $t_{CU1H}$   | $\overline{RES}$ Hold to CLK                              | 20                |      | ns    | (Note 1)                         |
| $t_{LIH}$    | Input Rise Time                                           |                   | 20   | ns    | From 0.8V to 2.0V                |
| $t_{LIL}$    | Input Fall Time                                           |                   | 12   | ns    | From 2.0V to 0.8V                |

**A.C. CHARACTERISTICS (Continued)****TIMING RESPONSES**

| Symbol                       | Parameter                          | Min. 8284A                    | Min. 8284A-1    | Max. | Units | Test Conditions   |
|------------------------------|------------------------------------|-------------------------------|-----------------|------|-------|-------------------|
| $t_{CLCL}$                   | CLK Cycle Period                   | 125                           | 100             |      | ns    |                   |
| $t_{CHCL}$                   | CLK HIGH Time                      | $(\frac{1}{2} t_{CLCL}) + 2$  | 39              |      | ns    |                   |
| $t_{CLCH}$                   | CLK LOW Time                       | $(\frac{1}{2} t_{CLCL}) - 15$ | 53              |      | ns    |                   |
| $t_{CH1CH2}$<br>$t_{CL2CL1}$ | CLK Rise or Fall Time              |                               |                 | 10   | ns    | 1.0V to 3.5V      |
| $t_{PHPL}$                   | PCLK HIGH Time                     | $t_{CLCL} - 20$               | $t_{CLCL} - 20$ |      | ns    |                   |
| $t_{PLPH}$                   | PCLK LOW Time                      | $t_{CLCL} - 20$               | $t_{CLCL} - 20$ |      | ns    |                   |
| $t_{RYLCL}$                  | Ready Inactive to CLK (See Note 3) | -8                            | -8              |      | ns    |                   |
| $t_{RYHCH}$                  | Ready Active to CLK (See Note 2)   | $(\frac{1}{2} t_{CLCL}) - 15$ | 53              |      | ns    |                   |
| $t_{CLIL}$                   | CLK to Reset Delay                 |                               |                 | 40   | ns    |                   |
| $t_{CLPH}$                   | CLK to PCLK HIGH DELAY             |                               |                 | 22   | ns    |                   |
| $t_{CLPL}$                   | CLK to PCLK LOW Delay              |                               |                 | 22   | ns    |                   |
| $t_{OLCH}$                   | OSC to CLK HIGH Delay              | -5                            | -5              | 22   | ns    |                   |
| $t_{OLCL}$                   | OSC to CLK LOW Delay               | 2                             | 2               | 35   | ns    |                   |
| $t_{OLOH}$                   | Output Rise Time (except CLK)      |                               |                 | 20   | ns    | From 0.6V to 2.0V |
| $t_{OHOL}$                   | Output Fall Time (except CLK)      |                               |                 | 12   | ns    | From 2.0V to 0.8V |

**NOTES:**

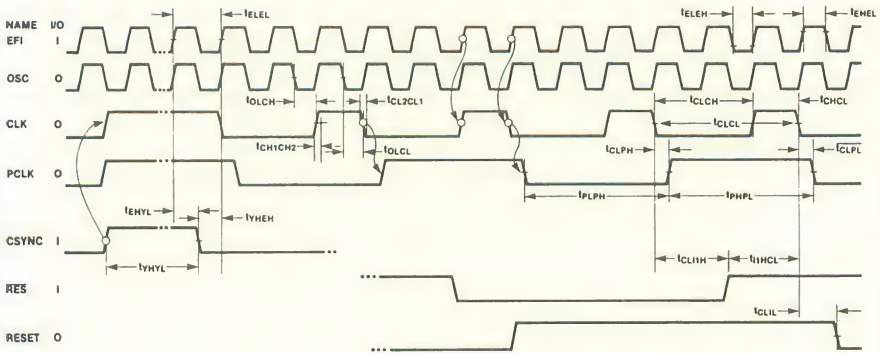
1. Setup and hold necessary only to guarantee recognition at next clock.
2. Applies only to T3 and TW states.
3. Applies only to T2 states.

**A.C. TESTING INPUT, OUTPUT WAVEFORM****A.C. TESTING LOAD CIRCUIT**



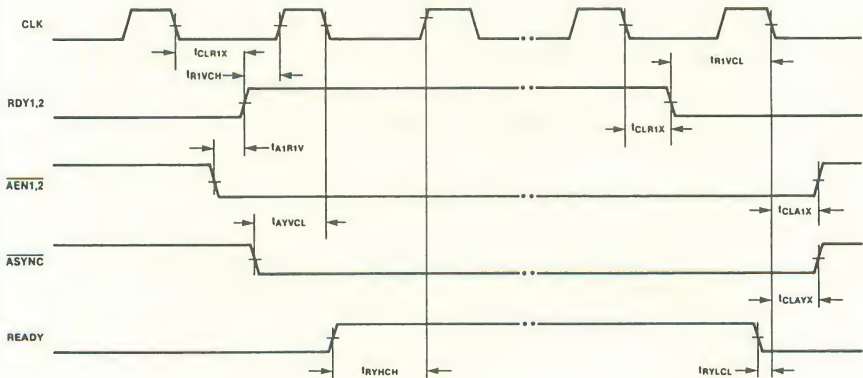
## WAVEFORMS

## CLOCKS AND RESET SIGNALS

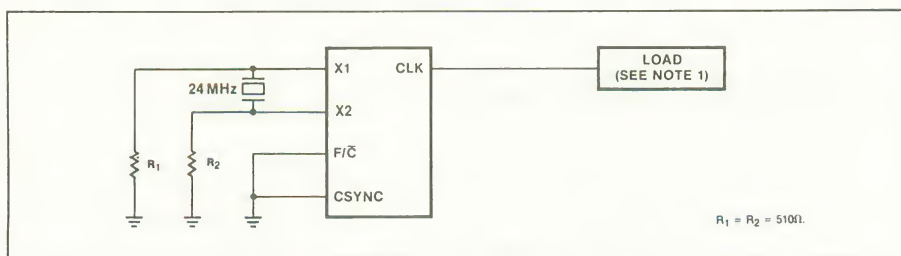
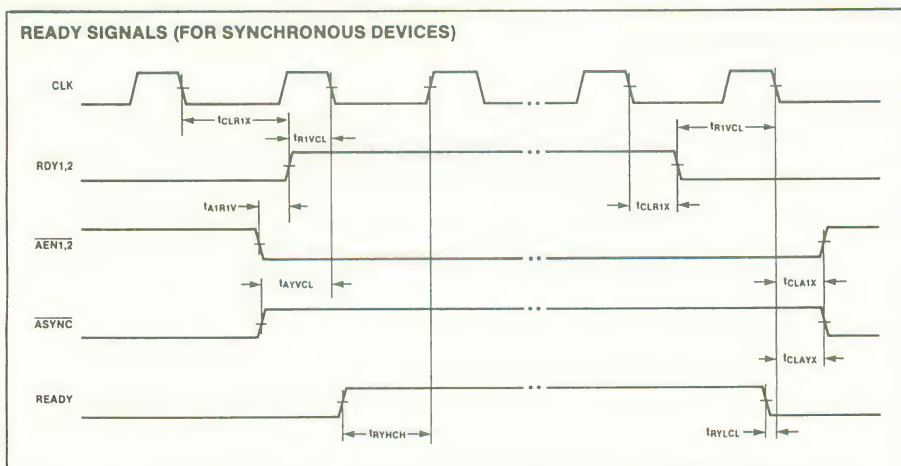


**NOTE: ALL TIMING MEASUREMENTS ARE MADE AT 1.5 VOLTS, UNLESS OTHERWISE NOTED.**

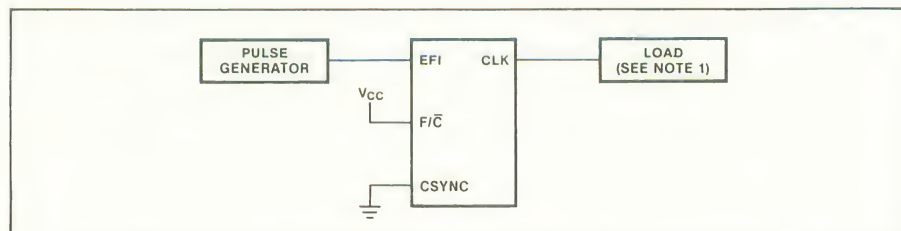
### READY SIGNALS (FOR ASYNCHRONOUS DEVICES)



## WAVEFORMS (Continued)

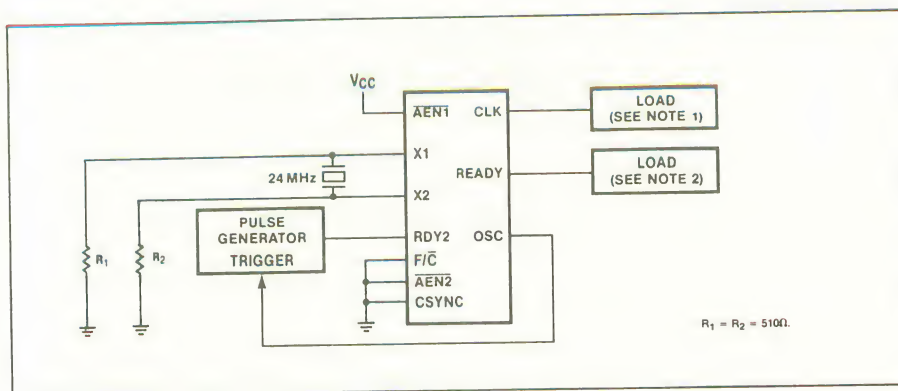


Clock High and Low Time (Using X1, X2)

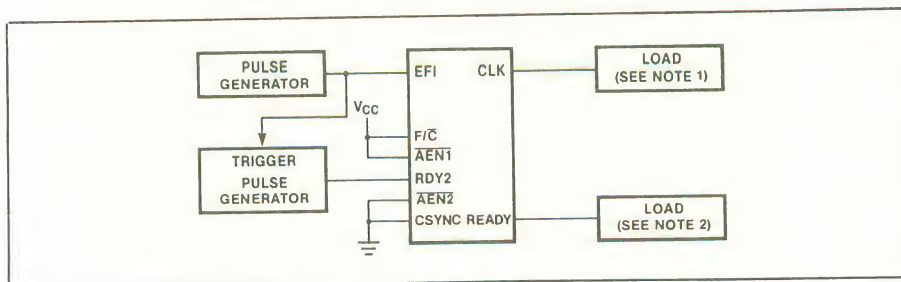


Clock High and Low Time (Using EFI)

## 8284A/8284A-1



Ready to Clock (Using X1, X2)



Ready to Clock (Using EFI)

## NOTES:

1.  $C_L = 100$  pF
2.  $C_L = 30$  pF

## 8286/8287 OCTAL BUS TRANSCEIVER

- Data Bus Buffer Driver for iAPX 86,88, MCS-80™, MCS-85™, and MCS-48™ Families
- High Output Drive Capability for Driving System Data Bus
- Fully Parallel 8-Bit Transceivers
- 3-State Outputs
- 20-Pin Package with 0.3" Center
- No Output Low Noise when Entering or Leaving High Impedance State

The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its outputs while the 8286 does not. Thus, a wide variety of applications for buffering in microcomputer systems can be met.

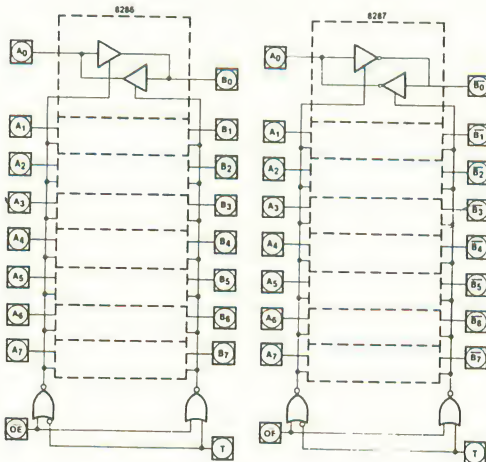


Figure 1. Logic Diagrams

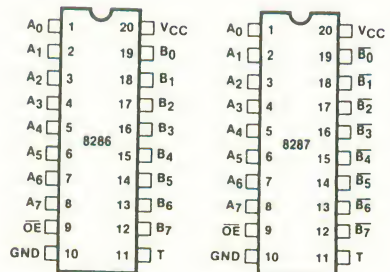
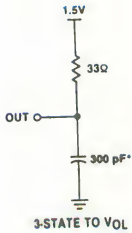
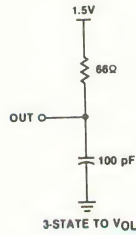


Figure 2. Pin Configurations

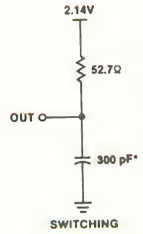
## TEST LOAD CIRCUITS



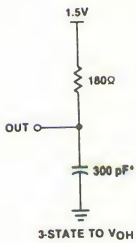
B OUTPUT



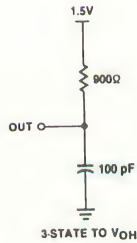
A OUTPUT



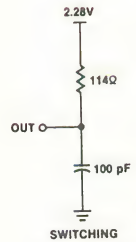
B OUTPUT



B OUTPUT



A OUTPUT



A OUTPUT

\*200 pF for plastic 8286/8287



**ABSOLUTE MAXIMUM RATINGS\***

Temperature Under Bias.....0°C to 70°C  
 Storage Temperature.....-65°C to +150°C  
 All Output and Supply Voltages.....-0.5V to +7V  
 All Input Voltages.....-1.0V to +5.5V  
 Power Dissipation.....1 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**D.C. CHARACTERISTICS** ( $V_{CC} = +5V \pm 10\%$ ,  $T_A = 0^\circ C$  to  $70^\circ C$ )

| Symbol                 | Parameter                                    | Min        | Max            | Units    | Test Conditions                                                         |
|------------------------|----------------------------------------------|------------|----------------|----------|-------------------------------------------------------------------------|
| $V_C$                  | Input Clamp Voltage                          |            | -1             | V        | $I_C = -5$ mA                                                           |
| $I_{CC}$               | Power Supply Current—8287<br>—8286           |            | 130<br>160     | mA<br>mA |                                                                         |
| $I_F$                  | Forward Input Current                        |            | -0.2           | mA       | $V_F = 0.45$ V                                                          |
| $I_R$                  | Reverse Input Current                        |            | 50             | $\mu$ A  | $V_R = 5.25$ V                                                          |
| $V_{OL}$               | Output Low Voltage —B Outputs<br>—A Outputs  |            | .45<br>.45     | V<br>V   | $I_{OL} = 32$ mA<br>$I_{OL} = 16$ mA                                    |
| $V_{OH}$               | Output High Voltage —B Outputs<br>—A Outputs | 2.4<br>2.4 |                | V<br>V   | $I_{OH} = -5$ mA<br>$I_{OH} = -1$ mA                                    |
| $I_{OFF}$<br>$I_{OFF}$ | Output Off Current<br>Output Off Current     |            | $I_F$<br>$I_R$ |          | $V_{OFF} = 0.45$ V<br>$V_{OFF} = 5.25$ V                                |
| $V_{IL}$               | Input Low Voltage —A Side<br>—B Side         |            | 0.8<br>0.9     | V<br>V   | $V_{CC} = 5.0$ V, See Note 1<br>$V_{CC} = 5.0$ V, See Note 1            |
| $V_{IH}$               | Input High Voltage                           | 2.0        |                | V        | $V_{CC} = 5.0$ V, See Note 1                                            |
| $C_{IN}$               | Input Capacitance                            |            | 12             | pF       | $F = 1$ MHz<br>$V_{BIAS} = 2.5$ V, $V_{CC} = 5$ V<br>$T_A = 25^\circ C$ |

**NOTE:**

1. B Outputs— $I_{OL} = 32$  mA,  $I_{OH} = -5$  mA,  $C_L = 300$  pF; A Outputs— $I_{OL} = 16$  mA,  $I_{OH} = -1$  mA,  $C_L = 100$  pF.

**A.C. CHARACTERISTICS** ( $V_{CC} = +5V \pm 10\%$ ,  $T_A = 0^\circ C$  to  $70^\circ C$ )

**Loading:** B Outputs— $I_{OL} = 32$  mA,  $I_{OH} = -5$  mA,  $C_L = 300$  pF  
 A Outputs— $I_{OL} = 16$  mA,  $I_{OH} = -1$  mA,  $C_L = 100$  pF

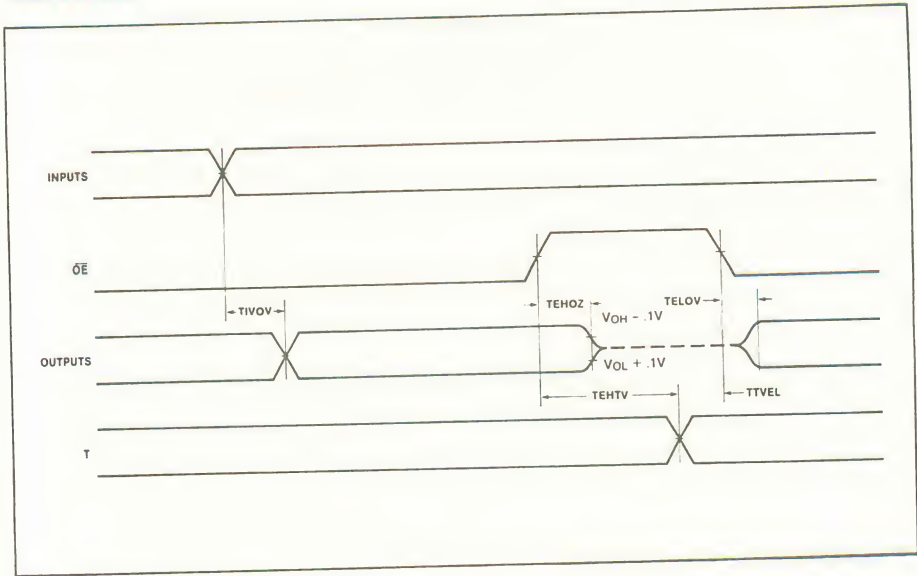
| Symbol          | Parameter                          | Min | Max | Units | Test Conditions    |
|-----------------|------------------------------------|-----|-----|-------|--------------------|
| TIVOV           | Input to Output Delay<br>Inverting | 5   | 22  | ns    | (See Note 1)       |
|                 | Non-Inverting                      | 5   | 30  | ns    |                    |
| TEHTV           | Transmit/Receive Hold Time         | 5   |     | ns    |                    |
| TTVEL           | Transmit/Receive Setup             | 10  |     | ns    |                    |
| TEHOZ           | Output Disable Time                | 5   | 18  | ns    |                    |
| TELOV           | Output Enable Time                 | 10  | 30  | ns    |                    |
| TILIH,<br>TOLOH | Input, Output Rise Time            |     | 20  | ns    | From 0.8 V to 2.0V |
| TIHIL,<br>TOHOL | Input, Output Fall Time            |     | 12  | ns    | From 2.0V to 8.0V  |

\* $C_L = 200$  pF for plastic 8286/8287

**NOTE:**

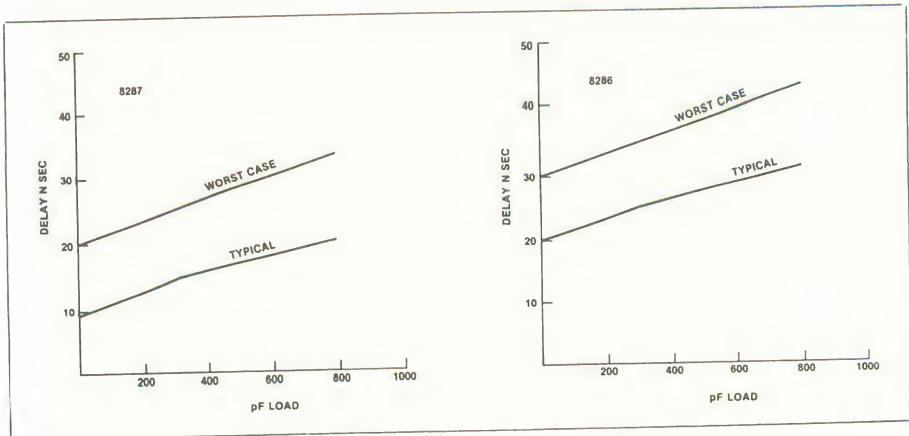
1. See waveforms and test load circuit on following page.

## WAVEFORMS



## NOTE:

1. All timing measurements are made at 1.5V unless otherwise noted.



Output Delay versus Capacitance

## 8288 BUS CONTROLLER FOR iAPX 86, 88 PROCESSORS

- Bipolar Drive Capability
- Provides Advanced Commands
- Provides Wide Flexibility in System Configurations
- 3-State Command Output Drivers
- Configurable for Use with an I/O Bus
- Facilitates Interface to One or Two Multi-Master Busses

The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large iAPX 86, 88 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

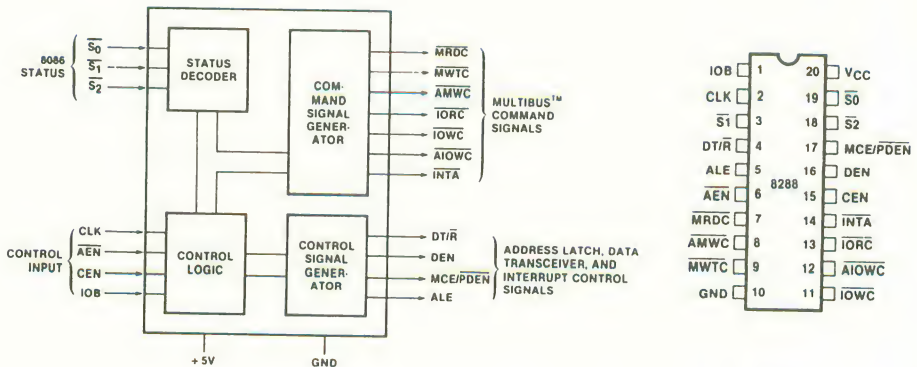


Figure 1. Block Diagram

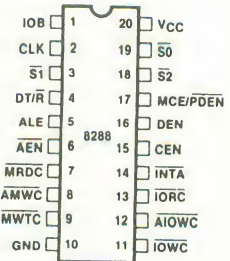


Figure 2.  
Pin Configuration

**ABSOLUTE MAXIMUM RATINGS\***

|                                |                 |
|--------------------------------|-----------------|
| Temperature Under Bias         | 0°C to 70°C     |
| Storage Temperature            | -65°C to +150°C |
| All Output and Supply Voltages | -0.5V to +7V    |
| All Input Voltages             | -1.0V to +5.5V  |
| Power Dissipation              | 1.5 Watt        |

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS** ( $V_{CC} = 5V \pm 10\%$ ,  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ )

| Symbol    | Parameter                                                 | Min. | Max. | Unit          | Test Conditions                        |
|-----------|-----------------------------------------------------------|------|------|---------------|----------------------------------------|
| $V_C$     | Input Clamp Voltage                                       |      | -1   | V             | $I_C = -5\text{ mA}$                   |
| $I_{CC}$  | Power Supply Current                                      |      | 230  | mA            |                                        |
| $I_F$     | Forward Input Current                                     |      | -0.7 | mA            | $V_F = 0.45\text{V}$                   |
| $I_R$     | Reverse Input Current                                     |      | 50   | $\mu\text{A}$ | $V_R = V_{CC}$                         |
| $V_{OL}$  | Output Low Voltage<br>Command Outputs<br>Control Outputs  |      | 0.5  | V             | $I_{OL} = 32\text{ mA}$                |
|           |                                                           |      | 0.5  | V             | $I_{OL} = 16\text{ mA}$                |
| $V_{OH}$  | Output High Voltage<br>Command Outputs<br>Control Outputs | 2.4  |      | V             | $I_{OH} = -5\text{ mA}$                |
|           |                                                           | 2.4  |      | V             | $I_{OH} = -1\text{ mA}$                |
| $V_{IL}$  | Input Low Voltage                                         |      | 0.8  | V             |                                        |
| $V_{IH}$  | Input High Voltage                                        | 2.0  |      | V             |                                        |
| $I_{OFF}$ | Output Off Current                                        |      | 100  | $\mu\text{A}$ | $V_{OFF} = 0.4\text{ to }5.25\text{V}$ |

**A.C. CHARACTERISTICS** ( $V_{CC} = 5V \pm 10\%$ ,  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ )**TIMING REQUIREMENTS**

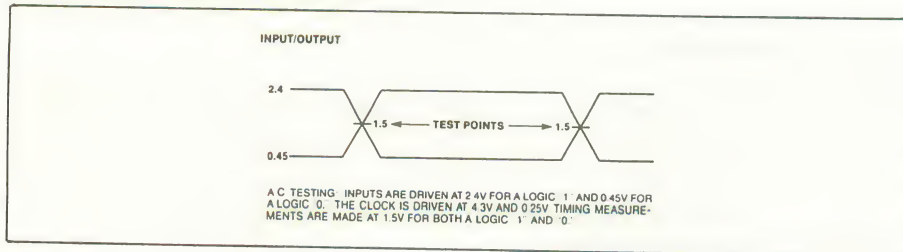
| Symbol | Parameter                  | Min. | Max. | Unit | Test Conditions   |
|--------|----------------------------|------|------|------|-------------------|
| TCLCL  | CLK Cycle Period           | 100  |      | ns   |                   |
| TCLCH  | CLK Low Time               | 50   |      | ns   |                   |
| TCHCL  | CLK High Time              | 30   |      | ns   |                   |
| TSVCH  | Status Active Setup Time   | 35   |      | ns   |                   |
| TCHSV  | Status Active Hold Time    | 10   |      | ns   |                   |
| TSHCL  | Status Inactive Setup Time | 35   |      | ns   |                   |
| TCLSH  | Status Inactive Hold Time  | 10   |      | ns   |                   |
| TILIH  | Input, Rise Time           |      | 20   | ns   | From 0.8V to 2.0V |
| TIHIL  | Input, Fall Time           |      | 12   | ns   | From 2.0V to 0.8V |

### A.C. CHARACTERISTICS (Continued)

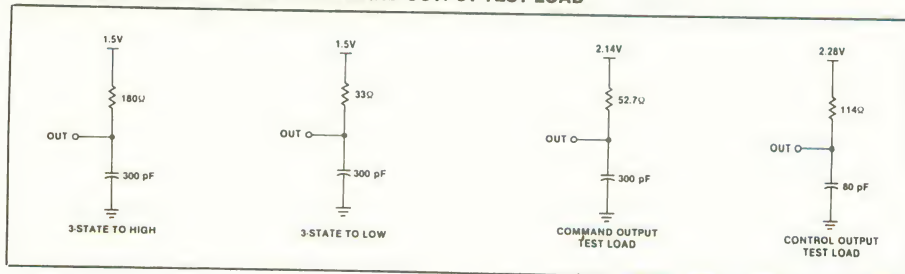
#### TIMING RESPONSES

| Symbol        | Parameter                          | Min. | Max.  | Unit | Test Conditions                                                                                                                                                                                                                                                                                                                                       |
|---------------|------------------------------------|------|-------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCVNV         | Control Active Delay               | 5    | 45    | ns   | <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> <math>I_{OL} = 32 \text{ mA}</math><br/> <math>I_{OH} = -5 \text{ mA}</math><br/> <math>C_L = 300 \text{ pF}</math> </div> <div> <math>I_{OL} = 16 \text{ mA}</math><br/> <math>I_{OH} = -1 \text{ mA}</math><br/> <math>C_L = 80 \text{ pF}</math> </div> </div> |
| TCVNX         | Control Inactive Delay             | 10   | 45    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCLLH, TCLMCH | ALE MCE Active Delay (from CLK)    |      | 20    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TSVLH, TSVMCH | ALE MCE Active Delay (from Status) |      | 20    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCHLL         | ALE Inactive Delay                 | 4    | 15    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCLML         | Command Active Delay               | 10   | 35    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCLMH         | Command Inactive Delay             | 10   | 35    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCHDTL        | Direction Control Active Delay     |      | 50    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCHDTH        | Direction Control Inactive Delay   |      | 30    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TAECH         | Command Enable Time                |      | 40    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TAHCZ         | Command Disable Time               |      | 40    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TAELCV        | Enable Delay Time                  | 115  | 200   | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TAENVV        | AEN to DEN                         |      | 20    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCEVVV        | CEN to DEN, PDEN                   |      | 25    | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TCELRH        | CEN to Command                     |      | TCLML | ns   |                                                                                                                                                                                                                                                                                                                                                       |
| TOLOH         | Output, Rise Time                  |      | 20    | ns   | From 0.8V to 2.0V                                                                                                                                                                                                                                                                                                                                     |
| TOHOL         | Output, Fall Time                  |      | 12    | ns   | From 2.0V to 0.8V                                                                                                                                                                                                                                                                                                                                     |

### A.C. TESTING INPUT, OUTPUT WAVEFORM

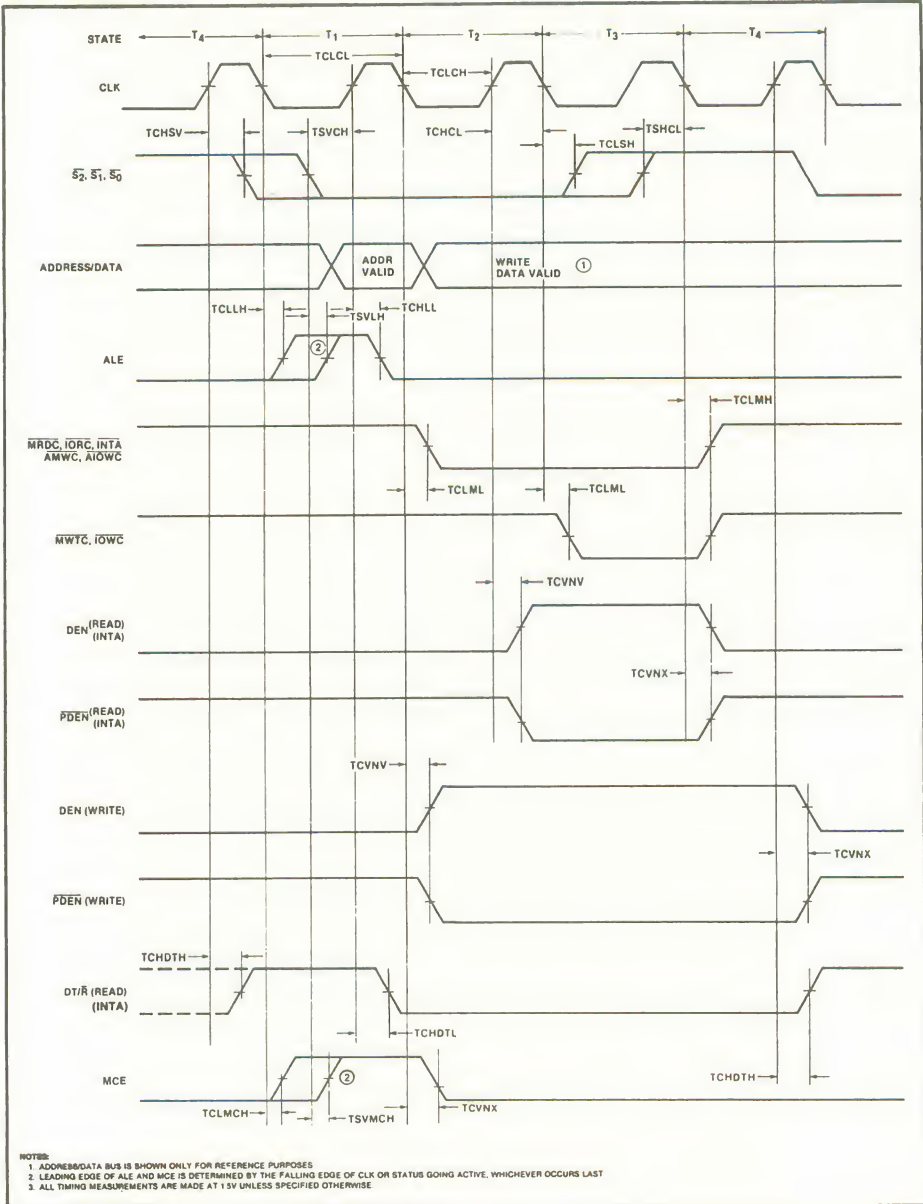


### TEST LOAD CIRCUITS—3-STATE COMMAND OUTPUT TEST LOAD



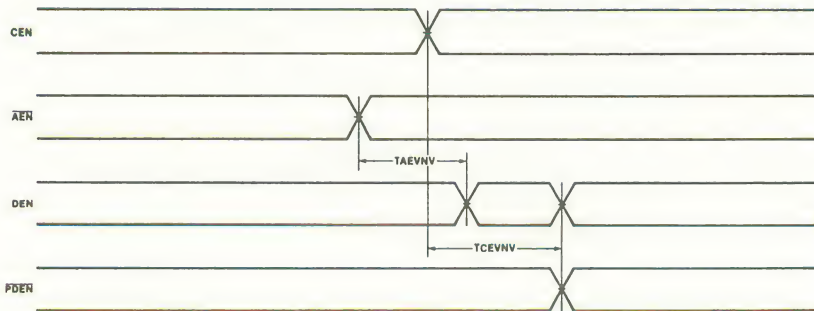


## WAVEFORMS

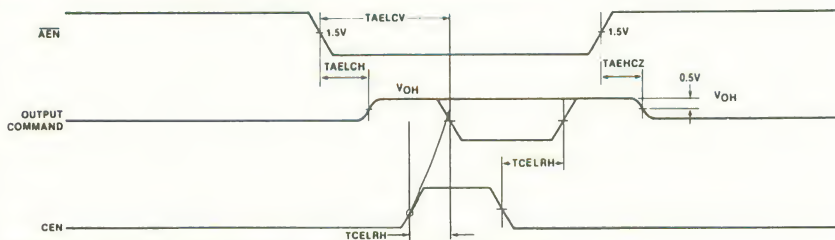


### WAVEFORMS (Continued)

### DEN, PDEN QUALIFICATION TIMING



### ADDRESS ENABLE (AEN) TIMING (3-STATE ENABLE/DISABLE)

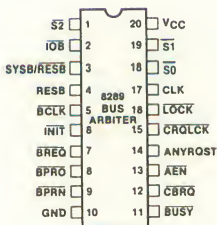


**NOTE:** CEN must be low or valid prior to T2 to prevent the command from being generated.

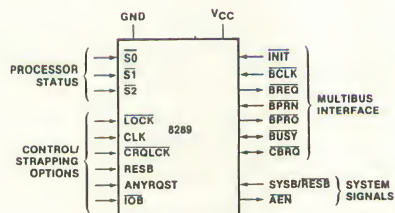
- Provides Multi-Master System Bus Protocol
- Synchronizes iAPX 86, 88 Processors with Multi-Master Bus
- Provides Simple Interface with 8288 Bus Controller
- Four Operating Modes for Flexible System Configuration
- Compatible with Intel Bus Standard MULTIBUS™
- Provides System Bus Arbitration for 8089 IOP in Remote Mode

The diagram illustrates the internal architecture of the 8086/8085/8088 STATUS interface. It is a self-contained module with a +5V supply and GND connection. The module's inputs are categorized into three groups: **8086/8085/8088 STATUS** (S<sub>2</sub>, S<sub>1</sub>, S<sub>0</sub>), **PROCESSOR CONTROL** (LOCK, CLK, CROWLCK, RESB, ANYRQST, IOB), and **MULTIBUS™ COMMAND SIGNALS** (INIT, BCLK, BREO, BPRN, BPRO, BUSY, CBRQ). Internally, these inputs feed into four functional blocks: **ARBITRATION**, **STATE GENERATOR**, **CONTROL**, and **MULTIBUS INTERFACE**. The **STATE GENERATOR** and **CONTROL** blocks are interconnected with bidirectional data buses. The **CONTROL** block also interfaces with the **LOCAL BUS INTERFACE**. The **MULTIBUS INTERFACE** block manages the communication with the external MULTIBUS™ system. The module's outputs include **AEN** and **SYSB/RESB**, which are part of the **SYSTEM SIGNALS**.

### Figure 1. Block Diagram



### Figure 2. Pin Diagram



### Figure 3. Functional Pinout

**ABSOLUTE MAXIMUM RATINGS\***

Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 All Output and Supply Voltages ..... -0.5V to +7V  
 All Input Voltages ..... -1.0V to +5.5V  
 Power Dissipation ..... 1.5 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = +5V \pm 10\%$ )

| Symbol       | Parameter                         | Min.           | Max.  | Units         | Test Condition                                                                |
|--------------|-----------------------------------|----------------|-------|---------------|-------------------------------------------------------------------------------|
| $V_C$        | Input Clamp Voltage               |                | - 1.0 | V             | $V_{CC} = 4.50V$ , $I_C = -5\text{ mA}$                                       |
| $I_F$        | Input Forward Current             |                | - 0.5 | mA            | $V_{CC} = 5.50V$ , $V_F = 0.45V$                                              |
| $I_R$        | Reverse Input Leakage Current     |                | 60    | $\mu\text{A}$ | $V_{CC} = 5.50$ , $V_R = 5.50$                                                |
| $V_{OL}$     | Output Low Voltage<br>BUSY, CBRQ  |                | 0.45  | V             | $I_{OL} = 20\text{ mA}$<br>$I_{OL} = 16\text{ mA}$<br>$I_{OL} = 10\text{ mA}$ |
|              | AEN                               |                | 0.45  | V             |                                                                               |
|              | BPRO, BREQ                        |                | 0.45  | V             |                                                                               |
| $V_{OH}$     | Output High Voltage<br>BUSY, CBRQ | Open Collector |       |               | $I_{OH} = 400\text{ }\mu\text{A}$                                             |
|              | All Other Outputs                 | 2.4            |       | V             |                                                                               |
| $I_{CC}$     | Power Supply Current              |                | 165   | mA            |                                                                               |
| $V_{IL}$     | Input Low Voltage                 |                | .8    | V             |                                                                               |
| $V_{IH}$     | Input High Voltage                | 2.0            |       | V             |                                                                               |
| Cin Status   | Input Capacitance                 |                | 25    | pF            |                                                                               |
| Cin (Others) | Input Capacitance                 |                | 12    | pF            |                                                                               |

**A.C. CHARACTERISTICS** ( $V_{CC} = +5V \pm 10\%$ ,  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ )**TIMING REQUIREMENTS**

| Symbol | Parameter                                  | Min.                 | Max.       | Unit | Test Condition    |
|--------|--------------------------------------------|----------------------|------------|------|-------------------|
| TCLCL  | CLK Cycle Period                           | 125                  |            | ns   |                   |
| TCLCH  | CLK Low Time                               | 65                   |            | ns   |                   |
| TCHCL  | CLK High Time                              | 35                   |            | ns   |                   |
| TSVCH  | Status Active Setup                        | 65                   | TCLCL-10   | ns   |                   |
| TSHCL  | Status Inactive Setup                      | 50                   | TCLCL-10   | ns   |                   |
| THVCH  | Status Active Hold                         | 10                   |            | ns   |                   |
| THVCL  | Status Inactive Hold                       | 10                   |            | ns   |                   |
| TBYSBL | BUSY $\uparrow$ Setup to BCLK $\downarrow$ | 20                   |            | ns   |                   |
| TCBSBL | CBRQ $\uparrow$ Setup to BCLK $\downarrow$ | 20                   |            | ns   |                   |
| TBLBL  | BCLK Cycle Time                            | 100                  |            | ns   |                   |
| TBHCL  | BCLK High Time                             | 30                   | .65[TBLBL] | ns   |                   |
| TCLLL1 | LOCK Inactive Hold                         | 10                   |            | ns   |                   |
| TCLLL2 | LOCK Active Setup                          | 40                   |            | ns   |                   |
| TPNBL  | BPRN $\uparrow$ to BCLK Setup Time         | 15                   |            | ns   |                   |
| TCLSR1 | SYSB/RESB Setup                            | 0                    |            | ns   |                   |
| TCLSR2 | SYSB/RESB Hold                             | 20                   |            | ns   |                   |
| TIVH   | Initialization Pulse Width                 | 3 TBLBL +<br>3 TCLCL |            | ns   |                   |
| TILIH  | Input Rise Time                            |                      | 20         | ns   | From 0.8 to 2.0V  |
| TIHIL  | Input Fall Time                            |                      | 12         | ns   | From 2.0V to 0.8V |

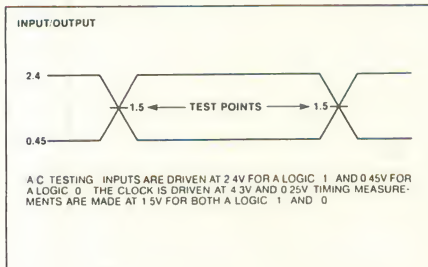
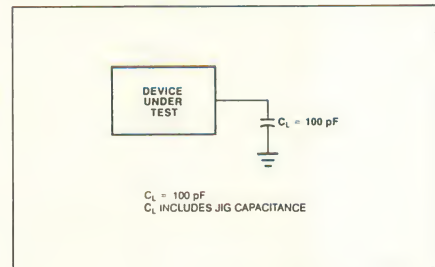
**A.C. CHARACTERISTICS (Continued)****TIMING RESPONSES**

| Symbol | Parameter                              | Min. | Max. | Unit | Test Condition    |
|--------|----------------------------------------|------|------|------|-------------------|
| TBLBRL | BCLK to BREQ Delay↓↑                   |      | 35   | ns   |                   |
| TBLPOH | BCLK to BPRO↓↑ (See Note 1)            |      | 40   | ns   |                   |
| TPNPO  | BPRN↓↑ to BPRO↓↑ Delay<br>(See Note 1) |      | 25   | ns   |                   |
| TBLBYL | BCLK to BUSY Low                       |      | 60   | ns   |                   |
| TBLBYH | BCLK to BUSY Float (See Note 2)        |      | 35   | ns   |                   |
| TCLAEH | CLK to AEN High                        |      | 65   | ns   |                   |
| TBLAEL | BCLK to AEN Low                        |      | 40   | ns   |                   |
| TBLCBL | BCLK to CBRQ Low                       |      | 60   | ns   |                   |
| TRLCRH | BCLK to CBRQ Float (See Note 2)        |      | 35   | ns   |                   |
| TOLOH  | Output Rise Time                       |      | 20   | ns   | From 0.8V to 2.0V |
| TOHOL  | Output Fall Time                       |      | 12   | ns   | From 2.0V to 0.8V |

↓↑ Denotes that spec applies to both transitions of the signal.

**NOTES:**

1. BCLK generates the first BPRO wherein subsequent BPRO changes lower in the chain are generated through BPRON.
2. Measured at .5V above GND.

**A.C. TESTING INPUT, OUTPUT WAVEFORM****A.C. TESTING LOAD CIRCUIT**



## WAVEFORMS



The signals related to CLK are typical processor signals, and do not relate to the depicted sequence of events of the signals referenced to BCLK. The signals shown related to the BCLK represent a hypothetical sequence of events for illustration. Assume 3 bus arbiters of priorities 1, 2 and 3 configured in serial priority resolving scheme as shown in Figure 6. Assume arbiter #1 has the bus and is holding busy low. Arbiter #2 detects its processor wants the bus and pulls low BREQ#2. If BPRN#2 is high (as shown), arbiter #2 will pull low CBQO line. CBQO signals to the higher priority arbiter #1 that a lower priority arbiter wants the bus. [A higher priority arbiter would be granted BPRN when it makes the bus request rather than having to wait for another arbiter to release the bus through CBQO]. \* Arbiter #1 will relinquish the multi-master system bus when it enters a state not requiring it (see Table 1), by lowering its BPRN#1 (tied to BPRN#2) and releasing BUSY. Arbiter #2 now sees that it has priority from BPRN#2 being low and releases CBQO. As soon as BUSY signifies the bus is available (high), arbiter #2 pulls BUSY low on next falling edge of BCLK. Note that if arbiter #2 didn't want the bus at the time it received priority, it would pass priority to the next lower priority arbiter by lowering its BPRO #2 (TPNP0).

\*\*Note that even a higher priority arbiter which is acquiring the bus through  $\overline{\text{BPRN}}$  will momentarily drop  $\overline{\text{CBRO}}$  until it has acquired the bus.

## 8089

### 8 & 16-BIT HMOS I/O PROCESSOR

- **High Speed DMA Capabilities Including I/O to Memory, Memory to I/O, Memory to Memory, and I/O to I/O**
- **IAPX 86, 88 Compatible: Removes I/O Overhead from CPU in IAPX 86/11 or 88/11 Configuration**
- **Allows Mixed Interface of 8- & 16-Bit Peripherals, to 8- & 16-Bit Processor Busses**
- **1 Mbyte Addressability**
- **Memory Based Communication with CPU**
- **Supports LOCAL or REMOTE I/O Processing**
- **Flexible, Intelligent DMA Functions Including Translation, Search, Word Assembly/Disassembly**
- **MULTIBUS™ Compatible System Interface**

The Intel® 8089 is a revolutionary concept in microprocessor input/output processing. Packaged in a 40-pin DIP package, the 8089 is a high performance processor implemented in N-channel, depletion load silicon gate technology (HMOS). The 8089's instruction set and capabilities are optimized for high speed, flexible and efficient I/O handling. It allows easy interface of Intel's 16-bit iAPX 86 and 8-bit iAPX 88 microprocessors with 8- and 16-bit peripherals. In the REMOTE configuration, the 8089 bus is user definable allowing it to be compatible with any 8/16-bit Intel microprocessor, interfacing easily to the Intel multiprocessor system bus standard MULTIBUS™.

The 8089 performs the function of an intelligent DMA controller for the Intel iAPX 86, 88 family and with its processing power, can remove I/O overhead from the iAPX 86 or iAPX 88. It may operate completely in parallel with a CPU, giving dramatically improved performance in I/O intensive applications. The 8089 provides two I/O channels, each supporting a transfer rate up to 1.25 mbyte/sec at the standard clock frequency of 5 MHz. Memory based communication between the IOP and CPU enhances system flexibility and encourages software modularity, yielding more reliable, easier to develop systems.

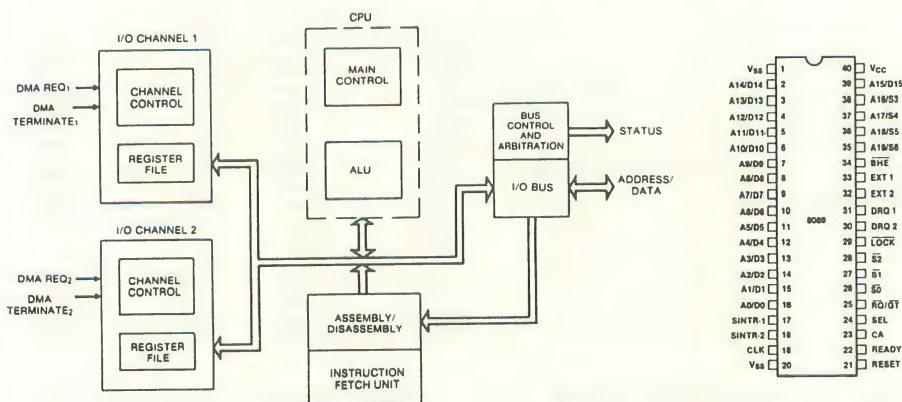


Figure 1. 8089 I/O Processor Block Diagram

Figure 2.  
8089 Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

## iAPX 86/20 iAPX 88/20 NUMERIC DATA PROCESSOR

- High Performance 2-Chip Numeric Data Processor
- Standard iAPX 86/10, 88/10 Instruction Set Plus Arithmetic, Trigonometric, Exponential, and Logarithmic Instructions For All Data Types
- All 24 iAPX 86/10, 88/10 Addressing Modes Available
- Conforms To Proposed IEEE Floating Point Standard
- Support 8 Data Types: 8-, 16-, 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands
- 8x80-Bit Individually Addressable Register Stack plus 14 General Purpose Registers
- 7 Built-in Exception Handling Functions
- MULTIBUS System Compatible Interface

The Intel iAPX 86/20 and iAPX 88/20 are two-chip numeric data processors (NDP's). They provide the instructions and data types needed for high-performance numeric applications. The NDP provides 100 times the performance of an iAPX 86/10, 88/10 CPU alone for numeric processing. The iAPX 86/20 consists of an iAPX 86/10 (16-bit 8086 CPU) and a numeric processor extension (NPX), the 8087. The iAPX 88/20 consists of the NPX in conjunction with the iAPX 88/10 (8-bit 8088 CPU). The NDP conforms to the proposed IEEE Floating Point Standard.

Both components of the iAPX 86/20 and iAPX 88/20 are implemented in N-channel, depletion load, silicon gate technology (HMOS), housed in two 40-pin packages. The iAPX 86/20, 88/20 adds 68 numeric processing instructions to the iAPX 86/10, 88/10 instruction set and eight 80-bit registers to the register set.

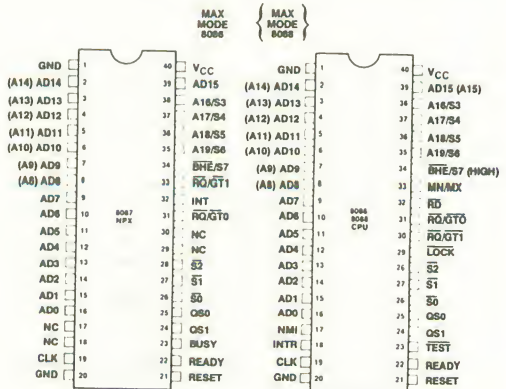
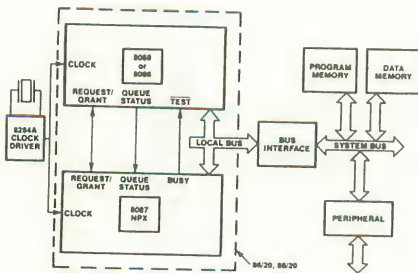


Figure 1. iAPX 86/20, 88/20 Block Diagram

Figure 2. iAPX 86/20, 88/20 Pin Configuration



# iAPX 86/30 iAPX 88/30 OPERATING SYSTEM PROCESSORS

80130-3

- High-Performance 2-Chip Data Processors Containing Operating System Primitives
- Standard iAPX 86/10, 88/10 Instruction Set Plus Task Management, Interrupt Management, Message Passing, Synchronization and Memory Allocation Primitives
- Fully Extendable To and Compatible With iRMX 86
- Supports Five Operating System Data Types: Jobs, Tasks, Segments, Mailboxes, Regions
- 35 Operating System Primitives
- Built-In Operating System Timers and Interrupt Control Logic Expandable From 8 to 57 Interrupts
- 8086/8087/8088 Compatible At Up To 8 MHz Without Wait States
- MULTIBUS System Compatible Interface

The Intel iAPX 86/30 and iAPX 88/30 are two-chip microprocessors offering general-purpose CPU (8086) instructions combined with real-time operating system support. They provide a foundation for multiprogramming and multitasking applications. The iAPX 86/30 consists of an iAPX 86/10 (16-bit 8086 CPU) and an Operating System Firmware (OSF) component (80130). The 88/30 consists of the OSF and an iAPX 88/10 (8-bit 8088 CPU).

Both components of the 86/30 and 88/30 are implemented in N-channel, depletion-load, silicon-gate technology (HMOS), and are housed in 40-pin packages. The 86/30 and 88/30 provide all the functions of the iAPX 86/10, 88/10 processors plus 35 operating system primitives, hardware support for eight interrupts, a system timer, a delay timer and a baud rate generator.

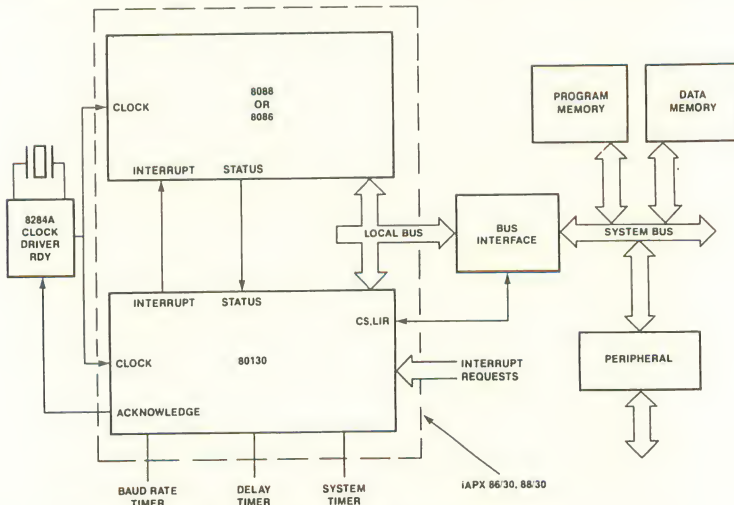


Figure 1. iAPX 86/30, 88/30 Block Diagram

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.  
OCTOBER 1981





# 付録D

## 8088 CPUについて

8088は、8ビット・データ・バスの8086マイクロプロセッサである。それ以外では、両者は同一である。したがって、以下の文では8088と8086の差について述べる。

### D.1 8088のプログラム可能レジスタとアドレッシング・モード

8088のプログラム可能レジスタとアドレッシング・モードは、すべてにおいて8086と同一である。実行速度を除いて、プログラマにとって8088は8086と同じである。

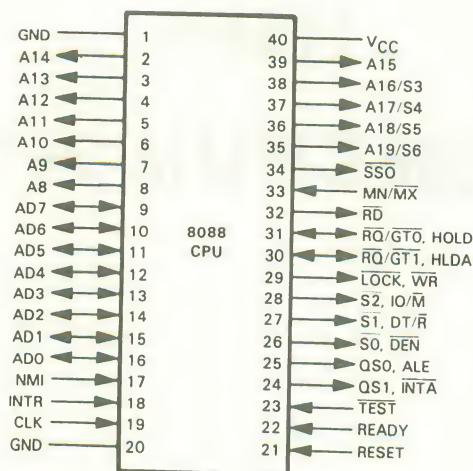
### D.2 8088CPUのピンと信号

8088CPUのピンと信号を図D-1に示す(次ページ)。図10-1に示されている8086のピンと信号との比較を行なうと、ピン34だけが異なる(ピン2-8と39がアドレスだけであることを除く)。

8086では、ピン34は $\overline{\text{BHE}}$ を出力する。この信号は、16ビットの8086データ・バス上の上位バイトと下位バイトとの間の区別を行なう。8088は8ビットのデータ・バスを有するので、 $\overline{\text{BHE}}$ と関連したロジックは無関係となる。8088は、マキシマム・モードの $\overline{\text{S0}}$ ステータスをピン34 ( $\overline{\text{SSO}}$ ) に出力する。

$\text{IO}/\overline{\text{M}}$ の信号は、8086と比べると、8088では反対の極性を持つ。これにより、8088は8085との互換性を有する。 $\text{IO}/\overline{\text{M}}$ 、 $\text{DT}/\overline{\text{R}}$ 、 $\overline{\text{SSO}}$ を組み合わせると、8088バス・サイクルは次のようにデコードされる。

| $\text{IO}/\overline{\text{M}}$ | $\text{DT}/\overline{\text{R}}$ | $\overline{\text{SSO}}$ |                |
|---------------------------------|---------------------------------|-------------------------|----------------|
| 0                               | 0                               | 0                       | コード・セグメント・アクセス |
| 0                               | 0                               | 1                       | メモリ・リード        |
| 0                               | 1                               | 0                       | メモリ・ライト        |
| 0                               | 1                               | 1                       | ノー・オペレーション     |
| 1                               | 0                               | 0                       | インタラプト・アクノリッジ  |
| 1                               | 0                               | 1                       | I/Oリード         |
| 1                               | 1                               | 0                       | I/Oライト         |
| 1                               | 1                               | 1                       | ホルト            |



| ピンの名前            | 種 類                     | 型            |
|------------------|-------------------------|--------------|
| AD0-AD 7         | アドレス/データ・バス             | 双方向, トライステート |
| A 8 - A15        | アドレス・バス                 | 出力, トライステート  |
| A16/S 3, A17/S 4 | アドレス/セグメント識別子           | 出力, トライステート  |
| A18/S 5          | アドレス/インタラプト・イネーブル・ステータス | 出力, トライステート  |
| A19/S 6          | アドレス/ステータス              | 出力, トライステート  |
| SS0              | ステータス出力                 | 出力, トライステート  |
| RD               | リード・コントロール              | 出力, トライステート  |
| READY            | ウェイト・ステート・リクエスト         | 出力, トライステート  |
| TEST             | テスト・コントロールのウェイト         | 入力           |
| INTR             | インタラプト・リクエスト            | 入力           |
| NMI              | ノンマスクابل・インタラプト・リクエスト   | 入力           |
| RESET            | システム・リセット               | 入力           |
| CLK              | システム・クロック               | 入力           |
| MN/MX            | マキシマム・システムではGND         |              |
| S0, S1, S2       | マシーン・サイクル・ステータス         | 出力, トライステート  |
| RQ/GT0, RQ/GT1   | ローカル・バス・プライオリティ・コントロール  | 双方向          |
| QS0, QS1         | インストラクション・キュー・ステータス     | 出力           |
| LOCK             | バス・ホールド・コントロール          | 出力, トライステート  |
| MN/MX            | ミニマム・システムではVcc          |              |
| IO/M             | メモリまたはI/Oのアクセス          | 出力, トライステート  |
| WR               | ライト・コントロール              | 出力, トライステート  |
| ALE              | アドレス・ラッチ・イネーブル          | 出力           |
| DT/R             | データ・トランスミット/レシーブ        | 出力, トライステート  |
| DEN              | データ・イネーブル               | 出力, トライステート  |
| INTA             | インタラプト・アクリノリッジ          | 出力, トライステート  |
| HOLD             | ホールド・リクエスト              | 出力, トライステート  |
| HLDA             | ホールド・アクリノリッジ            | 入力           |
| Vcc, GND         | パワー, グランド               | 出力           |

マキシマム・システムの信号

ミニマム・システムの信号

図D- 1 8088のピンと信号の割当て

8088はBHE信号を持たず、またそのような信号を必要としないので、8086についての外部メモリ・アドレッシングとBHEの議論は8088には適用されない。

### D.3 8088のタイミングと命令実行

8088は4バイトの命令オブジェクト・コードのキューを持つ。一方8086は、6バイトの命令オブジェクト・コードのキューを持つ。8088は、1バイトまたはそれ以上キューが空になるとただちに、4バイトのキューを満たすために、命令フェッチのバス・サイクルの実行を開始する。対照的に、8086は、6バイトのキューの2バイトまたはそれ以上が空になるまで、命令オブジェクト・コード・バイトのプリフェッチを開始しない。それ以外は、8086についてのバス・サイクルとキューのロジックの記述はそのまま8088に適用できる。

8088のキューは短いので、8086のキューではすべて含まれていた命令は、8088ではさらに命令のバイトを得るためにコードのフェッチを必要とする。各命令のフェッチには、4つのクロック・サイクルが付加されなければならない。たとえば、

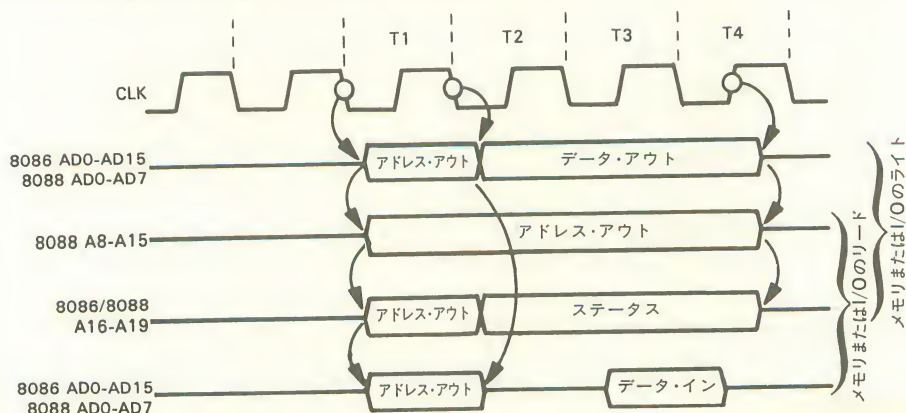
SUB TABLE [BX], 300

は、2バイトのディスプレイメント(TABLE)と2バイトのイミディエイト・データ(300)を含む6バイトの命令を表わす。最初の4バイトが8088の命令キューに含まれているとすると、イミディエイト・データはそれでもフェッチされなければならない。命令の実行時間に8つのクロック・サイクルが付加される。この規則は、8088の実行時間を8086の値から得るために適用される。

さらに、8088は8ビットのバスなので、8086が16ビットのデータをフェッチするために1つのバス・サイクルを実行していた場合は常に、2つのバス・サイクルが実行されなければならない。付録Aには、8086の実行時間が示されている。

### D.4 8088のメモリとI/O素子のアクセスのバス・サイクル

8088と8086のバス・サイクルのタイミングは、多重化データ／アドレス・バス・サイクルの点だけ異なっている。タイミングの相違は8つのアドレス・バス・ラインA8-A15

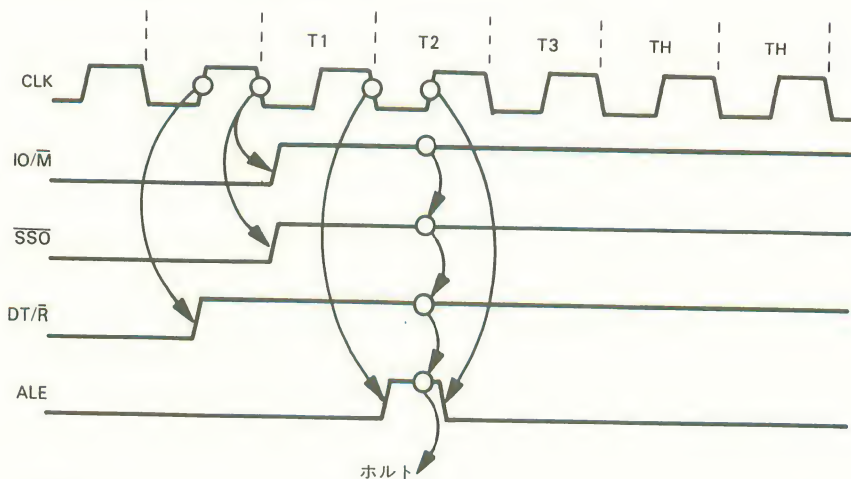


に限定され、前ページの図のように示される。

8088はBHE信号を持たない事実は別として、データ／アドレス・バス以外の信号のタイミングはすべて、8086と8088では同一である。

#### D.5 8088のホルト・ステート

ミニマム・モードでの動作は、8088のALEパルスは8086のタイミングと比較して1クロック期間だけ遅れる。これは次のように図示される。



それ以外、ホルト・ステートのロジックとタイミングは8086と8088で同一である。

#### D.6 8086と互換性のある8088の他のロジック

8086と8088のロジックは、次のステートとロジックで完全に同一である。

1. ウェート・ステート
2. ホールド・ステート
3.  $\overline{\text{RQ}}/\overline{\text{GT}}$ ロジック
4. ロックのロジック
5. テスト・ステートのウェート
6. プロセッサのエスケープ
7. デバイス・リセット
8. インタラプト処理
9. シングル・ステップ・モード

#### D.7 8088命令セット

この本の数多くの表に示されている、8086と8088の命令セットは、実行時間を除いて同一である。

# 欧 文 索 引

## [A]

|                                                                    |                     |
|--------------------------------------------------------------------|---------------------|
| AAA (ASCII Adjust for Addition) .....                              | 47, 66, 281         |
| $\overline{\text{AACK}}$ (advanced acknowledge) .....              | 454                 |
| AAD (ASCII Adjust for Division) .....                              | 49, 67, 290         |
| AAM (ASCII Adjust for Multiplication) .....                        | 49, 69, 286         |
| AAS (ASCII Adjust for Subtraction) .....                           | 47, 70, 284         |
| $\overline{\text{ADA0}} - \overline{\text{ADR13}}$ (address) ..... | 451                 |
| ADC (Add with Carry) .....                                         | 47, 72, 73, 75, 280 |
| ADD (Add) .....                                                    | 47, 77, 79, 81, 280 |
| AND (AND) .....                                                    | 48, 82, 84, 85, 296 |
| ASCIIストリングの和 (sum a pair of multibyte ASCII strings) .....         | 282                 |
| ASCIIによる乗算 (ASCII multiplication) .....                            | 287                 |
| ASCIIによる除算 (ASCII division) .....                                  | 288                 |

## [B]

|                                                     |                   |
|-----------------------------------------------------|-------------------|
| BCD数値の和 (sum a pair of multibyte BCD numbers) ..... | 279               |
| $\overline{\text{BCLK}}$ (bus clock) .....          | 452               |
| $\overline{\text{BHE}}$ (Bus High Enable) .....     | 360, 370          |
| $\overline{\text{BHEN}}$ (bus high enable) .....    | 452               |
| BIU (Bus Interface Unit) .....                      | 26, 383, 384, 397 |
| $\overline{\text{BPRN}}$ (bus priority in) .....    | 452               |
| $\overline{\text{BPRO}}$ (bus priority out) .....   | 453               |
| $\overline{\text{BREQ}}$ (bus request) .....        | 453               |
| $\overline{\text{BUSY}}$ (bus busy) .....           | 453               |

## [C]

|                                                     |                         |
|-----------------------------------------------------|-------------------------|
| CALL (CALL) .....                                   | 47, 87, 89, 90, 92, 307 |
| CAS (cascade address) .....                         | 434                     |
| $\overline{\text{CBRQ}}$ (common bus request) ..... | 453                     |
| CBW (Convert Byte to Word) .....                    | 47, 93, 290             |
| $\overline{\text{CCLK}}$ (constant clock) .....     | 452                     |



|                                                   |                       |
|---------------------------------------------------|-----------------------|
| CLC (Clear Carry flag) .....                      | 49, 95, 315           |
| CLD (Clear Direction flag) .....                  | 49, 96, 315           |
| CLI (Clear Interrupt flag) .....                  | 49, 97, 315           |
| CLK (clock) .....                                 | 397, 400              |
| CMC (Complement Carry flag) .....                 | 49, 98, 315           |
| CMP (Compare) .....                               | 47, 99, 100, 102, 292 |
| CMPS (Compare String) .....                       | 47, 105, 303          |
| CPU (Central Processing Unit) .....               | 2                     |
| CPUの状態の退避 (saving the state of the machine) ..... | 277                   |
| CWD (Convert Word to Double) .....                | 47, 107, 290          |

## [D]

|                                        |                   |
|----------------------------------------|-------------------|
| DAA (Decimal Adjust Addition) .....    | 48, 108, 281      |
| DAS (Decimal Adjust Subtraction) ..... | 48, 109, 284      |
| DAT0 — DATF (data) .....               | 452               |
| DEC (Decrement) .....                  | 47, 111, 112, 283 |
| DIV (Divide) .....                     | 49, 114, 289      |

## [E]

|                           |              |
|---------------------------|--------------|
| EOF (End-of-File) .....   | 17, 21, 22   |
| ESC (Escape) .....        | 47, 116, 315 |
| ESCAPE (ESC参照) .....      | 392, 463     |
| EU (Execution Unit) ..... | 26, 383, 384 |

## [H]

|                                                   |                   |
|---------------------------------------------------|-------------------|
| 8086 .....                                        | 357               |
| 8088 .....                                        | 543               |
| 8251 (Programmable Communication Interface) ..... | 29, 353           |
| 8257 (DMA Controller) .....                       | 438               |
| 8259 (Priority Interrupt Controller) .....        | 422, 425          |
| 8282/8283 (8-bit Bistable Latches) .....          | 367               |
| 8284 (Clock Generator/Driver) .....               | 28, 398           |
| 8286/8287 (Octal Transceivers) .....              | 376               |
| 8288 (Buss Controller) .....                      | 28, 389, 394, 466 |
| 8289 (Bus Arbiter) .....                          | 391, 466, 475     |
| HLT (Halt) .....                                  | 47, 117, 315      |
| HOLD/HLDA (hold/hold acknowledge) .....           | 435               |

## [ I ]

|                                                                     |                   |
|---------------------------------------------------------------------|-------------------|
| IDIV (Integer Divide).....                                          | 49, 119, 289      |
| IMUL (Integer Multiply).....                                        | 48, 121, 286      |
| IN (Input).....                                                     | 47, 123, 124, 317 |
| INC (Increment).....                                                | 47, 125, 127, 280 |
| $\overline{\text{INH1}}, \overline{\text{INH2}}$ (inhibit) .....    | 452               |
| INIT (initialize) .....                                             | 451               |
| INT (Interrupt) .....                                               | 49, 129, 319      |
| $\overline{\text{INTA}}$ (interrupt acknowledge) .....              | 454               |
| INTO (Interrupt if Overflow) .....                                  | 49, 130, 319      |
| $\overline{\text{INT0}} - \overline{\text{INT7}}$ (interrupt) ..... | 454               |
| $\overline{\text{IORC}}$ (I/O read control) .....                   | 453               |
| $\overline{\text{IOWC}}$ (I/O write control) .....                  | 454               |
| IRET (Interrupt Return) .....                                       | 49, 132, 319      |

## [ J ]

|                                                       |                                  |
|-------------------------------------------------------|----------------------------------|
| JA (Jump if Above) .....                              | 47, 133, 311                     |
| JAE (Jump if Above or Equal) .....                    | 47, 133, 311                     |
| JB (Jump if Below) .....                              | 47, 134, 311                     |
| JBE (Jump if Below or Equal) .....                    | 47, 135, 311                     |
| JC (Jump if Carry) $\Rightarrow$ JB                   |                                  |
| JCXZ (Jump if CX is Zero) .....                       | 47, 135, 312                     |
| JE (Jump if Equal) .....                              | 47, 136, 311                     |
| JG (Jump if Greater) .....                            | 47, 137, 311                     |
| JGE (Jump if Greater or Equal) .....                  | 47, 138, 311                     |
| JL (Jump if Less) .....                               | 47, 138, 311                     |
| JLE (Jump if Less or Equal) .....                     | 47, 139, 311                     |
| JMP (Jump) .....                                      | 47, 140, 141, 143, 144, 146, 308 |
| JNA (Jump if Not Above) $\Rightarrow$ JBE             |                                  |
| JNAE (Jump if Not Above nor Equal) $\Rightarrow$ JB   |                                  |
| JNB (Jump if Not Below) $\Rightarrow$ JAE             |                                  |
| JNBE (Jump if Not Below nor Equal) $\Rightarrow$ JA   |                                  |
| JNC (Jump if Not Carry) $\Rightarrow$ JAE             |                                  |
| JNE (Jump if Not Equal) .....                         | 47, 147, 312                     |
| JNG (Jump if Not Greater) $\Rightarrow$ JLE           |                                  |
| JNGE (Jump if Not Greater nor Equal) $\Rightarrow$ JL |                                  |
| JNL (Jump if Not Less) $\Rightarrow$ JGE              |                                  |

|                                                    |              |
|----------------------------------------------------|--------------|
| JNLE (Jump if Not Less nor Equal) $\Rightarrow$ JG |              |
| JNO (Jump if Not Overflow) .....                   | 47, 148, 312 |
| JNP (Jump if Not Parity) .....                     | 47, 149, 312 |
| JNS (Jump if Not Sign) .....                       | 47, 149, 312 |
| JNZ (Jump if Not Zero) $\Rightarrow$ JNE           |              |
| JO (Jump if Overflow) .....                        | 47, 150, 312 |
| JP (Jump if Parity) .....                          | 47, 151, 312 |
| JPE (Jump if Parity Even) $\Rightarrow$ JP         |              |
| JPO (Jump if Parity Odd) $\Rightarrow$ JNP         |              |
| JS (Jump if Sign) .....                            | 47, 151, 312 |
| JZ (Jump if Zero) $\Rightarrow$ JE                 |              |

## [L]

|                                                |                        |
|------------------------------------------------|------------------------|
| LAHF (Load AH Flags) .....                     | 47, 152, 268           |
| LDS (Load register and DS) .....               | 47, 154, 267           |
| LEA (Load Effective Address) .....             | 47, 155, 267           |
| LES (Load register and ES) .....               | 47, 157, 267           |
| LOCK (Lock) .....                              | 47, 158, 315, 396, 397 |
| LODS (Load String) .....                       | 47, 159, 303           |
| LOOP (Loop) .....                              | 47, 161, 314           |
| LOOPE (Loop if Equal) .....                    | 47, 162, 314           |
| LOOPNE (Loop if Not Equal) .....               | 47, 163, 314           |
| LOOPNZ (Loop if Not Zero) $\Rightarrow$ LOOPNE |                        |
| LOOPZ (Loop if Zero) $\Rightarrow$ LOOPE       |                        |

## [M · N]

|                                                       |                                            |
|-------------------------------------------------------|--------------------------------------------|
| MN/ $\overline{\text{MX}}$ .....                      | 357, 360, 389                              |
| MOV (Move) .....                                      | 47, 163, 165, 166, 168, 169, 171, 172, 266 |
| MOVS (Move String) .....                              | 47, 174, 303                               |
| $\overline{\text{MRDC}}$ (memory read control) .....  | 453                                        |
| MUL (Multiply) .....                                  | 48, 176, 286                               |
| $\overline{\text{MWTC}}$ (memory write control) ..... | 453                                        |
| NEG (Negate) .....                                    | 47, 177, 284                               |
| NOP (No Operation) .....                              | 47, 179, 315                               |
| NOT (NOT) .....                                       | 47, 180, 296                               |

## [O · P · Q]

|                              |     |
|------------------------------|-----|
| OFFSETオペレータ (operator) ..... | 345 |
|------------------------------|-----|

|                               |                        |
|-------------------------------|------------------------|
| OR (OR) .....                 | 48, 181, 182, 184, 296 |
| OUT (Output) .....            | 47, 186, 187, 317      |
| POP (Pop) .....               | 47, 189, 190, 192, 268 |
| POPF (Pop Flags).....         | 49, 193, 268           |
| PUSH (Push) .....             | 47, 195, 196, 198, 267 |
| PUSHF (Push Flags).....       | 47, 200, 267           |
| QS0, QS1 (queue status) ..... | 362, 390, 392          |

[ R ]

|                                                     |                             |
|-----------------------------------------------------|-----------------------------|
| RCL (Rotate through Carry Left) .....               | 48, 201, 322                |
| RCR (Rotate through Carry Right) .....              | 48, 203, 323                |
| REP (Repeat) .....                                  | 47, 205, 304                |
| RET (Return) .....                                  | 47, 206, 208, 209, 210, 307 |
| ROL (Rotate Left) .....                             | 48, 212, 324                |
| ROR (Rotate Right) .....                            | 48, 213, 325                |
| $\overline{RQ}/\overline{GT}$ (request/grant) ..... | 440                         |

[ S ]

|                                          |                        |
|------------------------------------------|------------------------|
| SAHF (Store AH into 8080 Flags).....     | 49, 215, 268           |
| SAL (Shift Left) $\Rightarrow$ SHL       |                        |
| SAR (Shift Arithmetic Right) .....       | 48, 217, 327           |
| SBB (Subtract with Borrow) .....         | 47, 219, 220, 222, 283 |
| SCAS (Scan String) .....                 | 47, 224, 303           |
| SEG (Segment) .....                      | 47, 226, 302           |
| SHL (Shift Left) .....                   | 48, 228, 326           |
| SHR (Shift Right) .....                  | 48, 231, 328           |
| Signetics オブジェクト・コード (object code) ..... | 298                    |
| STC (Set Carry flag) .....               | 49, 233, 315           |
| STD (Set Direction flag) .....           | 49, 234, 315           |
| STI (Set Interrupt flag) .....           | 49, 235, 315           |
| STOS (Store String) .....                | 47, 237, 303           |
| SUB (Subtract) .....                     | 47, 239, 240, 242, 283 |

[ T · W · X ]

|                                                |                        |
|------------------------------------------------|------------------------|
| Tステート (state) .....                            | 364                    |
| TEST (Test) .....                              | 48, 244, 246, 248, 297 |
| WAIT (Wait) .....                              | 47, 250, 315           |
| $\overline{XACK}$ (transfer acknowledge) ..... | 454                    |

|                          |                        |
|--------------------------|------------------------|
| XCHG (Exchange) .....    | 47, 251, 253, 266      |
| XLAT (Translate) .....   | 47, 254, 267           |
| XOR (Exclusive-OR) ..... | 48, 256, 257, 259, 297 |

## 和 文 索 引

### 〔ア 行〕

|                                              |               |
|----------------------------------------------|---------------|
| アイドル・クロック期間 (idle clock period: TI) .....    | 366, 388      |
| アセンブラ (assembler) .....                      | 3, 329, 338   |
| アセンブリ言語 (assembly language) .....            | 1, 338        |
| アドレス・バス (address bus) .....                  | 359, 367      |
| アドレッシング・テーブル (addressing table) .....        | 61            |
| アドレッシング・モード (addressing modes) .....         | 50            |
| アドレッシング・モード・バイト (addressing mode byte) ..... | 58            |
| アルゴリズム (algorithm) .....                     | 2             |
| 暗黙指定アドレッシング (implied addressing) .....       | 54            |
| イミディエイト・アドレッシング (immediate addressing) ..... | 52            |
| インタラプト (interrupt) .....                     | 413           |
| インタラプト・アクノリッジ (interrupt acknowledge) .....  | 416           |
| インタラプト・フラグ (Interrupt Flag: IF) .....        | 45            |
| インタラプト命令 (interrupt instructions) .....      | 318           |
| インダイレクト・アドレッシング (indirect addressing) .....  | 51            |
| インデックス・レジスタ (index registers) .....          | 43            |
| ウエート状態 (ウエート・ステート: wait state: TW) .....     | 360, 394, 408 |
| エグゼキューション・ユニット (Execution Unit: EU) .....    | 26, 383, 384  |
| エディタ (editor) .....                          | 329, 331      |
| オーバーフロー・フラグ (Overflow Flag: OF) .....        | 45            |
| オブジェクト・プログラム (object program) .....          | 2             |

### 〔カ 行〕

|                                          |               |
|------------------------------------------|---------------|
| カスケード・アドレス (cascade address: CAS) .....  | 434           |
| キー (key) .....                           | 18            |
| キャリー・フラグ (Carry Flag: CF) .....          | 45            |
| キュー・ステータス (queue status: QS1, QS2) ..... | 362, 390, 392 |



|                                      |          |
|--------------------------------------|----------|
| クロック (clock: CLK) .....              | 397, 400 |
| コプロセッサ (co-processor) .....          | 392, 463 |
| コントロール情報 (control information) ..... | 6, 7     |

## 〔サ 行〕

|                                                         |               |
|---------------------------------------------------------|---------------|
| 32ビットの乗算 (32-bit multiply) .....                        | 285           |
| サイン・フラグ (Sign Flag: SF) .....                           | 45            |
| 算術演算命令 (arithmetic instructions) .....                  | 279           |
| シェル・ソート (Shell sort) .....                              | 18            |
| シングル・ステップ (single step) .....                           | 10, 45, 415   |
| 出力イネーブル (Output Enable: OE) .....                       | 372, 374, 375 |
| スタック・アドレッシング (stack addressing) .....                   | 57            |
| ストリング・プリミティブ命令 (string primitive instructions) .....    | 302           |
| セグメント・レジスタ (segment registers) .....                    | 43            |
| セグメント・レジスタの初期設定 (segment register initialization) ..... | 278           |
| セグメント変更 (segment override) .....                        | 60            |
| セグメント変更プレフィックス (segment override prefixes) .....        | 302           |
| ゼロ・フラグ (Zero Flag: ZF) .....                            | 45            |
| ソース・プログラム (source program) .....                        | 3             |
| ソフトウェア・インタラプト (software interrupt) .....                | 416           |
| 疎結合 (loosely coupled) .....                             | 459, 466      |

## 〔タ 行〕

|                                                         |               |
|---------------------------------------------------------|---------------|
| ダイレクト・アドレッシング (direct addressing) .....                 | 51, 53        |
| ダイレクト・インデックス修飾アドレッシング (direct indexed addressing) ..... | 54            |
| 多重化バス (multiplexed bus) .....                           | 359, 364      |
| データ・バス (data bus) .....                                 | 359, 367, 372 |
| データ・メモリ (data memory) .....                             | 2             |
| データ移動命令 (data movement instruction) .....               | 265           |
| ディジー・チェーン (daisy chain) .....                           | 453, 457      |
| ディレクション・フラグ (Direction Flag: DF) .....                  | 45            |
| デバッガ (debugger) .....                                   | 10, 329, 340  |
| デバッグ (debug) .....                                      | 10            |
| 定義済インタラプト (predefined interrupts) .....                 | 414           |
| トラップ・フラグ (Trap Flag: TF) .....                          | 45            |
| 動作モード (operating mode) .....                            | 389           |

## 〔ナ 行〕

|                                                     |          |
|-----------------------------------------------------|----------|
| 入出力命令 (I/O instructions) .....                      | 316      |
| ノンマスカブル・インタラプト (non-maskable interrupt : NMI) ..... | 361, 415 |

## 〔ハ 行〕

|                                                                               |                         |
|-------------------------------------------------------------------------------|-------------------------|
| ハードウェア・インタラプト (hardware interrupt) .....                                      | 416                     |
| パラメータの受け渡し (parameter passing) .....                                          | 29, 351                 |
| パリティ・フラグ (Parity Flag : PF) .....                                             | 45                      |
| バス・インターフェイス・ユニット (Bus Interface Unit : BIU) ...                               | 26, 383, 384, 397       |
| バス・コントロール (bus control) .....                                                 | 435                     |
| バス・サイクル (bus cycle) .....                                                     | 364                     |
| バス・スレーブ (bus slave) .....                                                     | 449                     |
| バス・マスタ (bus master) .....                                                     | 449                     |
| バスのタイミング (bus timing) .....                                                   | 426, 427, 431           |
| バス競合 (bus contention) .....                                                   | 377, 379, 382, 383, 452 |
| バッファからバッファへの移動 (buffer-to-buffer move) .....                                  | 269                     |
| バッファの変換 (translating a buffer) .....                                          | 275, 294                |
| 汎用レジスタ (general purpose registers) .....                                      | 42                      |
| フラグ・レジスタ (flag register) .....                                                | 44                      |
| プログラム・カウンタ制御命令 (program counter control instructions) .....                   | 306                     |
| プログラム・ステータス・ワード (Program Status Word : PSW)                                   |                         |
| ⇨フラグ・レジスタ                                                                     |                         |
| プログラム・メモリ (program memory) .....                                              | 2                       |
| プログラム設計 (program design) .....                                                | 8                       |
| プログラム相対アドレッシング (program relative addressing) .....                            | 51                      |
| プロセッサ制御命令 (processor control instructions) .....                              | 310                     |
| ブレイクポイント (breakpoint) .....                                                   | 416                     |
| ブロック・コントロール・キャラクタ (Block Control Character) .....                             | 298                     |
| 複数ワード数値の和 (sum a pair of multiword numbers) .....                             | 279                     |
| ベース相対アドレッシング (base relative addressing) .....                                 | 28, 55                  |
| ベース相対ダイレクト・インデックス修飾アドレッシング (base relative direct<br>indexed addressing) ..... | 57                      |
| ベース相対ダイレクト・スタック・アドレッシング (base relative direct stack<br>addressing) .....      | 57                      |
| ベース相対暗黙指定アドレッシング (base relative implied addressing) .....                     | 57                      |
| ホールド・アクノリッジ (hold acknowledge) .....                                          | 359, 360                |
| ポインタ・レジスタ (pointer registers) .....                                           | 43                      |

|                                             |      |
|---------------------------------------------|------|
| ポート (port) .....                            | 6, 7 |
| 補助キャリー・フラグ (Auxiliary carry Flag: AF) ..... | 45   |

## 〔マ行〕

|                                  |                    |
|----------------------------------|--------------------|
| マキシマム・モード (maximum mode) .....   | 361, 389, 449, 463 |
| マルチバス (Multibus) .....           | 449                |
| マルチプロセッサ (multiprocessor) .....  | 463                |
| ミニマム・モード (minimum mode) .....    | 361, 389, 449      |
| 密結合 (tightly coupled) .....      | 459, 466           |
| メンテナンス (maintenace) .....        | 13                 |
| 命令キュー (instruction queue) .....  | 362, 384           |
| 命令フェッチ (instruction fetch) ..... | 383                |
| モジュール (module) .....             | 8                  |

## 〔ラ・ヤ行〕

|                                                         |     |
|---------------------------------------------------------|-----|
| リエントラント・プログラム (再入可能プログラム: re-entrant program) .....     | 28  |
| リセット (reset) .....                                      | 404 |
| リロケイト可能 (再配置可能: relocatable) .....                      | 28  |
| レディ (ready) .....                                       | 408 |
| 64ビットの除算 (64-bit division) .....                        | 291 |
| ローテートとシフトの命令 (rotate and shift instructions) .....      | 320 |
| 論理演算命令 (logical instructions) .....                     | 294 |
| 有効メモリ・アドレス (実効メモリ・アドレス: effective memory address) ..... | 50  |

<著者紹介>

ラッセル・レクター (Russell Rector)

1968年より計算機処理に携わる。カリフォルニア大学よりコンピュータ・サイエンスの学士号を受けた後、いくつかの重要なソフトウェア・システムの創作に参加し、後にOsborneのテクニカル・スタッフに加わり、ソフトウェア設計とOsborneの出版物の著作協力で時間をさいている。

ジョージ・アレクシー (George Alexy)

1977年にIntelに入る。8086、8088、8089、8087を含む8ビットと16ビットのマイクロプロセッサをカバーする、マイクロプロセッサの製品のアプリケーション・マネージャである。彼のグループは、シングルとマルチのプロセッサ・システムに対するシステム設計法、CPUとシステムの構成と性能に関連するリソース分配と機能的な分割に関係している。

<訳者紹介>

吉川敏則 (よしかわ としのり)

昭和46年東京工業大学工学部電子工学科卒業／昭和51年同大学院博士課程修了。工学博士／昭和52年埼玉大学工学部電子工学科講師／現在、長岡技術科学大学助教授、中央大学非常勤講師／専門はデジタル信号処理、回路解析、コンピュータのソフトウェア応用。

主な著書：「実用BASIC—科学技術計算」(昭晃堂)、「最新マイコン・ガイド」(日本理工出版会)、「16ビットパソコン・ガイド」(日本理工出版会)、「工学における数値計算法」(日本理工出版会)、「PC-8801実用数値計算プログラム集」(昭晃堂)、「PC-9801実用数値計算プログラム集」(昭晃堂)、「BASICノート」(日本理工出版会)、「PC-9801E/F/M16ビットアセンブラ入門」(産業報知センター)

ザ8086ブック

定価4,800円

1982年9月20日 初版発行

1985年5月10日 9版発行

1986年1月25日 新装版発行

著者 R. レクター  
G. アレクシー  
訳者 吉川敏則  
発行者 三好哲雄  
発行所 秋葉出版株式会社

東京都千代田区神田和泉町1-1(郵便番号101)

<検印廃止>

電話 03 (866) 5491 代表 振替 東京7-161626

印刷=株式会社廣済堂

製本=秋元製本株式会社

© Printed in Japan, 1982

万一乱丁・落丁がございましたら、書店または発行所でおとりかえいたします。

ISBN4-87184-040-9











# ザ8086ブック

16ビット・マイクロプロセッサ  
8086・8088の使い方

ラッセル・レクター ジョージ・アレクシー 共著 吉川敏則 訳

## 16ビットの決定版8086

16ビット・マイクロプロセッサの本格的な応用は今後ますます盛んになるだろう。これまでの8ビット汎用マイクロプロセッサにない高度な機能は、新しい世代のマイクロコンピュータの製品を生み出す有力な武器を提供している。

定価4,800円

ISBN4-87184-040-9 C3055 ¥4800E